MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Kozák Jan** Personal ID number: **420381**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Branch of study: **Data Science**

II. Master's thesis details

Master's thesis title in English:

Efficient Algorithms for Relational Marginal Polytope Construction

Master's thesis title in Czech:

Efektivní algoritmy pro konstrukci relačních marginálních polytopů

Guidelines:

1. Get familiar with the framework of Markov logic networks and the probabilistic inference problems tackled within it.
2. Get familiar with the primal formulation of Markov logic network learning (so-called "relational marginal problems") and with the notion of "relational marginal polytope".
3. Design efficient heuristic algorithms for construction of relational marginal polytopes. Consider also approximation algorithms with rigorous guarantees.
4. Implement the designed algorithms and use them either inside the recent method for weight learning of Markov logic networks (using the algorithm described in Kuželka, O. and Kungurtsev, V., AISTATS 2019) or for removing redundant first-order logic formulas from Markov logic networks.
5. Compare your algorithms to the naive domain-lifted algorithms based on reductions from weighted-first-order model counting from (Kuzelka, O., and Wang, Y., AISTATS 2020)
6. Optional: Can you design a faster algorithm for detecting when the relational marginal polytope lives in a lower dimensional affine subspace (i.e. when some of the formulas are redundant) instead of constructing the polytope?

Bibliography / sources:

[1] Koller, D., Friedman, N., Džeroski, S., Sutton, C., McCallum, A., Pfeffer, A., ... & Neville, J. (2007). Introduction to statistical relational learning. MIT press.

[2] Richardson, Matthew, and Pedro Domingos. 'Markov logic networks.' Machine learning 62.1-2 (2006): 107-136.

[3] Kuželka, O., Wang, Y., Davis, J., & Schockaert, S. Relational marginal problems: Theory and estimation. In Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[4] Kuželka, O. and Kungurtsev, V., Lifted Weight Learning of Markov Logic Networks Revisited. AISTATS 2019: 22nd International Conference on Artificial Intelligence and Statistics, 2019.
[5] Kuželka, O., Wang Y., Domain-Liftability of Relational Marginal Polytopes, AISTATS 2019: 23rd International Conference on Artificial Intelligence and Statistics, 2020.

Name and workplace of master's thesis supervisor:

Ing. Ondřej Kuželka, Ph.D., Intelligent Data Analysis, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **11.02.2020** Deadline for master's thesis submission: **21.05.2021**

Assignment valid until: **30.09.2021**

Ing. Ondřej Kuželka, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

F3

Faculty of Electrical Engineering
Department of Computer Science

Master's Thesis

Efficient Algorithms for Relational Marginal Polytope Construction

Jan Kozák

May 2021
Supervisor: Ing. Ondřej Kuželka, Ph.D.

Acknowledgement / Declaration

I would like to thank my supervisor, my family and my girlfriend for their support.

I declare that this thesis has been composed solely by myself and except wherestates otherwise by reference or acknowledgment, the work presented is entirely my own.

In Prague, 21 May 2021

.....

Abstrakt / Abstract

Cílem práce je navržení efektivních heuristických a/nebo aproximačních algoritmů pro konstrukci *relačních marginálních polytopů*. Ty jsou geometrickou reprezentací množiny přípustných řešení tzv. relačního marginálního problému, což je konvexní optimalizační úloha hledající pravděpodobnostní rozdělení nad množinou možných světů, které splňuje zadané marginální pravděpodobnosti formulí v Markovské logické síti a dosahuje maximální entropie. Navržený algoritmus je porovnán s naivním exaktním doménově liftovatelným algoritmem popsaným Kuželkou a Wangem v jejich článku *Domain-Liftability of Relational Marginal Polytopes*, 2020 [1].

Klíčová slova: Markovské logické sítě, relační marginální polytopy

Překlad titulu: Efektivní algoritmy pro konstrukci relačních marginálních polytopů

The goal of the thesis is to design efficient heuristic and/or approximation algorithms for construction of *relational marginal polytopes*, a geometrical representation of the set of feasible solutions of the relational marginal problem, which is a convex optimization task searching the max-entropy distribution over possible worlds satisfying defined marginal probabilities for formulas in a Markov logic network. The designed algorithm is compared to naive exact domain-liftable algorithm described by Kuželka and Wang in their article *Domain-Liftability of Relational Marginal Polytopes*, 2020 [1].

Keywords: Markov Logic Networks, Relational Marginal Polytopes

/ Contents

Tables / Figures

Chapter 1
Introduction

Markov logic networks (MLN) are relatively novel type of probabilistic logic — a broad field of models and formalisms that combines methods of probability theory for handling the uncertainty with methods of standard propositional or first-order logic to infer new information from a knowledge bases of formulas. MLNs are formed by tuples of first-order logic formulas and their weights and may be interpreted as a softened version of the first-order logic. The main topic of the thesis — *relational marginal polytopes* — emerge from so called called *relational marginal problems* in the MLNs. Goal of these tasks is finding the max-entropy distribution over possible worlds of the MLN satisfying defined marginal probabilities over the formulas. The relational marginal polytope represents the set of feasible solutions to this task and is also important for solving the dual problem of maximum-likelihood learning of the formula weights of the MLN. Markov logic networks may be also considered a template for creation of Markov random fields (also called Markov networks), which are — together with Bayesian networks — one of the most commonly used probabilistic graphical models.

The thesis is structured in the following way:

- First, in Preliminaries chapter, the basic concepts related to the whole area of first-order logic, probabilistic logic and probabilistic graphical models are defined and explained,
- in the next chapter Markov logic networks and its important properties will be defined and described together with important algorithms,
- finally in the Implementation chapter the approaches to constructing relational marginal polytopes will be discussed and evaluated.

Chapter 2
Preliminaries

This chapter provides a basic background about mathemathical, logical and machine learning concepts that are related to the topic of the thesis. First the first-order logic (FOL) considered in the thesis is described, followed by description of probabilistic logics which incorporate uncertainty into the standard first-order or propositonal logics. Finally a notion of probabilistic graphical models is debated, focused on Bayesian networks and Markov random fields. The latter are integral part of Markov logic networks, the key topic of the thesis.

2.1 First-Order Logic

The thesis considers a function-free first-order logic language $\mathcal{L}$ built from sets $Const$ (constants), $Vars$ (variables) and Rel (predicates). The set of predicates Rel is partitioned into subsets Rel_i each containing predicates of arity i, so $Rel = \bigcup_i Rel_i$. The constants represent the domain objects (e.g. `Alice`, `Bob`, `Prague`) and the variable symbols range over them. The predicates represent relations among objects (e.g. `friends`) or their attributes (e.g. `capital`). These three sets together constitute *non-logical symbols* and their actual meaning is specified by an *interpretation*. In addition to them the language $\mathcal{L}$ is also built from a standard set of *logical symbols*:

- *universal* ($\forall$) and *existential* ($\exists$) quantifiers,
- unary logical connective – *negation* ($\neg$),
- binary logical connectives – *and* ($\wedge$), *or* ($\vee$), *implication* ($\Rightarrow$) and *equivalence* ($\Leftrightarrow$).

First-order logic theories about domains being modelled are formulated by means of *formulas*. Following list summarizes terminology related to their creation.

- **Term** is a constant or a variable.
- **Atom** or **atomic formula** is a k-ary predicate $R(a_1, a_2, ..., a_k)$ with arguments $a_1, a_2, ...a_k \in Const \cup Vars$ (i.e. terms).
- **Literal** is an atom or its negation.
- **Formula** is a literal or a logical connection of two formulas (may be also applied recursively),
  - set of variables appearing in formula α is denoted as $Vars(\alpha)$,
  - formula α is called *ground formula* if its arguments are constants,
  - formula α_0 is called *grounding of formula* α if it can be obtained by substituting all variables in $Vars(\alpha)$ with constants from $Const$,
  - a variable in a formula is called *free* if it is not bound by any quantifier.
- **Sentence** is a formula with no free variables.

A special type of formula is a *clause* which is a disjunction of literals. Every formula in FOL can be mechanically transformed to conjunction of clauses, so called *clausal form* or *conjuctive normal form* (CNF). This form is convenient for automated processing

and due to beforementioned transformation we can consider all formulas to be in CNF without loss of generality.

A possible world ω is an assignment of truth values to every possible ground atom. A formula is *satisfiable* if there exists at least one possible world in which it holds true. All formulas together form a *knowledge base* (KB). The knowledge base might be considered a one big conjunction of all its formulas, as in basic setting it is expected that all formulas in the KB are simultaneously true. A typical inference problem involving usage of a knowledge base is to decide if the KB *entails* formula F (denoted as $KB \models F$), that is if F is true whenever KB holds. This is usually checked by *refutation* – $KB \models F$ holds iff $KB \cup \neg F$ is not satisfiable. Note however that this yields a positive answer also in cases when KB contains a contradiction.

First-order logic used in the thesis is further restricted by following assumptions:

- unique names assumption – different constants refer to different objects,
- injective substitution – different variables in a formula must be mapped to different terms,
- only domains of finite size are considered.

2.1.1 Probabilistic logic

Probabilistic logic is an extension of standard predicate (or propositional) logic which aims to handle uncertainty about actual truth values of formulas. Most common ways to achieve this goal are either specifying a probability that the formula is true or using multi-valued logic. An example of the former approach is the probabilistic logic defined in (Nilsson, 1986 [2]), which is the basis for formalism used in Markov Logic Networks, the main topic of the thesis. The latter approach is usually described in terms of *fuzzy logic* where the truth value of a formula may be any real number in interval [0,1].

The key difference between these two concepts is that in the (Nilsson's) probabilistic logic it is assumed the formula is true with some probability (let's say 0.5), but in the end the formula will eventually be evaluated as strictly true or false. The probability just captures our *belief* about the actual truth value — we are not sure what the value is at first, but once we are, there's no room for any value between true and false and the probabilistic logic becomes a standard 0–1 valued predicate logic. On the other hand it is perfectly valid to state that a truth value of a formula is 0.5 in fuzzy logic as it is built upon fuzzy set theory which extends the set membership function from bivalent to multi-valued, usually being defined as real number in the unit interval (but fuzzy theories with discrete values are also studied) [3].

With multi-valued logic it's possible to formally capture vague or imprecise definitions that naturally arise in everyday language, such as "*Tom is a little old.*" This may be represented as a predicate *old(Tom)*. In the standard predicate logic, we would have to decide if *a little old* is enough to declare this predicate true (maybe after asking for Tom's exact age and comparing it with some threshold), but in fuzzy logic the truth value of *old(Tom)* may be set to some appropriate value such as 0.3, indicating that Tom is not "fully" old yet but he's indeed a little old. With extending the range of possible truth values we also need to redefine behaviour of logic connectives (usually conjunction and implication) and it turns out there is not just one unique way how to do it, but there are actually many well-behaved definitions, each one creating a slightly different variant of fuzzy logic. Examples of some commonly used fuzzy conjunctions are shown in Figure 2.1.

The probabilistic logic as defined by Nilsson introduces a *probability of sentence* and *possible worlds* semantics to incorporate uncertainty about the truth values into the

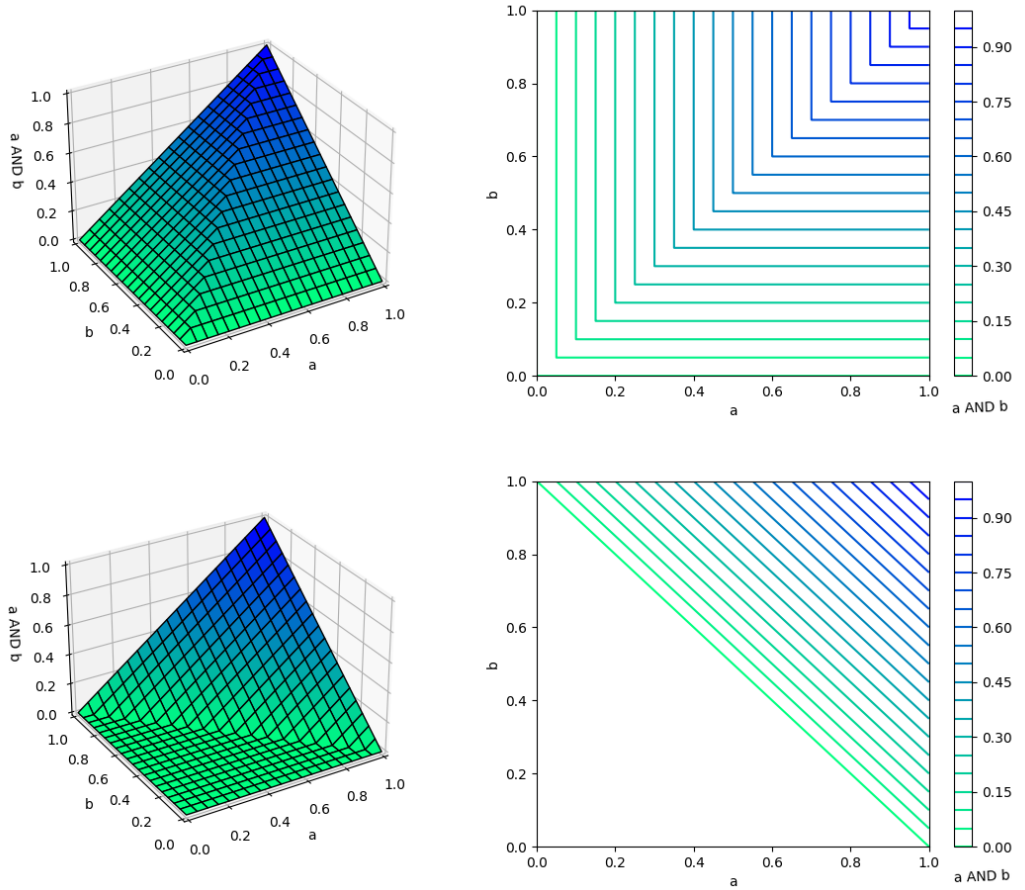

Figure 2.1. Surface and contour plots of two fuzzy conjunction examples which are also *triangular norms* (t-norms). **Upper**: Minimum t-norm $\top_{min} = \min\{a, b\}$. **Lower**: Łukasiewicz t-norm $\top_{Luk} = \max\{0, a + b - 1\}$.

first-order logic. If we consider only one sentence S, the sentence may be either *true* or *false*. This induces two sets of possible worlds — $\mathcal{W}_1$ containing possible worlds where S is true and $\mathcal{W}_2$ containing the worlds where S is false. Then we can reason about the truth value of sentence S in terms of probabilities by specifiying probability p_1 that the actual world is in $\mathcal{W}_1$ (and S is therefore true) and probability $p_2 = 1 - p_1$ that the actual world is in $\mathcal{W}_2$. We can then say that the *(probabilistic) truth value* of sentence S is p_1.

When we incorporate more sentences, the number of sets of possible worlds rises as every set of possible worlds $\mathcal{W}_i$ now represents a distinct combination of truth values assigned to each sentence. For N sentences this may result in up to 2^N sets of possible worlds, but usually their total count is lower as some combinations are logically inconsistent and therefore define an *impossible world* (e.g. S_1 true, S_2 true but $S_3 = S_1 \wedge S_2$ false). The set of consistent possible worlds is then considered a sample space over which a probability distribution is defined. For every set of possible worlds $\mathcal{W}_i$ a probability p_i specifies the probability that the actual world is in $\mathcal{W}_i$. As the sets of possible worlds are exclusive and exhaustive, all p_i sum to 1. The probabilistic truth value of a sentence S is then simply defined as a sum of probabilities of all sets of possible worlds where S is true. Analogically the logical entailment of sentence S from set of sentences $\mathcal{B}$ ($\mathcal{B} \vdash S$) is generalized as the *probabilistic entailment* which is the probability that S is true given the probabilities of sentences in $\mathcal{B}$ (set of beliefs).

Now suppose there are N sentences $S_1, S_2, ...S_N$ which together specify K sets of consistent possible worlds, denote the probabilistic truth values of sentences as a column vector $\Pi = [\pi_1, \pi_2, ..., \pi_N]$, denote the probability distribution over the possible worlds as $P = [p_1, p_2, ...p_K]$ and denote the actual truth values of sentences associated with each possible world as matrix V of dimensions $N \times K$, where element v_{ij} represents the truth value of sentence S_i in set of possible worlds $\mathcal{W}_j$. Note that each column of V there represents one set of possible worlds. Calculation of the probabilistic truth values of all sentences then may be concisely represented as a matrix equation

$$\Pi = VP \tag{2.1}$$

As a concrete example consider a theory with three sentences (taken from Nilsson's original article [2])

- $S_1 = \forall x : \ P(x)$,
- $S_2 = \forall x : \ P(x) \Rightarrow Q(x)$,
- $S_3 = \forall x : \ Q(x)$.

The sentences define 4 distinct sets of possible worlds with following combinations of consistent truth values:

	$\mathcal{W}_1$	$\mathcal{W}_2$	$\mathcal{W}_3$	$\mathcal{W}_4$
$S_1 = \forall x : \ P(x)$	*true*	*true*	*false*	*false*
$S_2 = \forall x : \ P(x) \Rightarrow Q(x)$	*true*	*false*	*true*	*true*
$S_3 = \forall x : \ Q(x)$	*true*	*false*	*true*	*false*

Table 2.1. Consistent combinations of truth values of sentences in possible worlds.

Translation of the table to matrix V is straightforward and omitted. Instead we'll focus on possible range of the truth values π_i. As we see from Equation (2.1), the value of Π depends on probabilities of possible worlds P. Now consider at first the extremal case where exactly one possible world achieves probability 1 and the probability of the rest is 0. This obviously results in Π being equal to the column of V corresponding with the currently selected set of posible worlds. We can then proceed with modifying probabilities p_i which in turn changes the outcome of all π_i. The probabilities p_i are however also constrained as their sum must be 1, so the actual attainable truth values pi_i are convex combinations of those achieved for extremal distributions of p_i. This is visualized in Figure 2.2. In this geometrical interpretation the extremal values are vertices of a polytope and all attainable truth values of the sentences lie inside or on boundaries of the polytope.

Figure 2.2 also shows that it is not straightforward to just arbitrarily set values of π_i independently on each other, as their consistent combinations are restricted by the polytope. This doesn't pose a problem in case when the calculation proceeds exactly in the direction of Equation (2.1) and the probability distribution of possible worlds is already specified, because the equation guarantees the result Π will be consistent. In practice however the reasoning often works the other way around — the probabilities of some sentences are assigned first (e.g. as an input from some expert), the sentences then form the knowledge base, and the goal is to find probabilities of the other sentences, i.e. to evaluate a probabilistic entailment of the sentences with unspecified probabilities from those in the knowledge base. In this setting actual probability values P of possible worlds may not be even specified in advance as we're just interested in the values of Π.

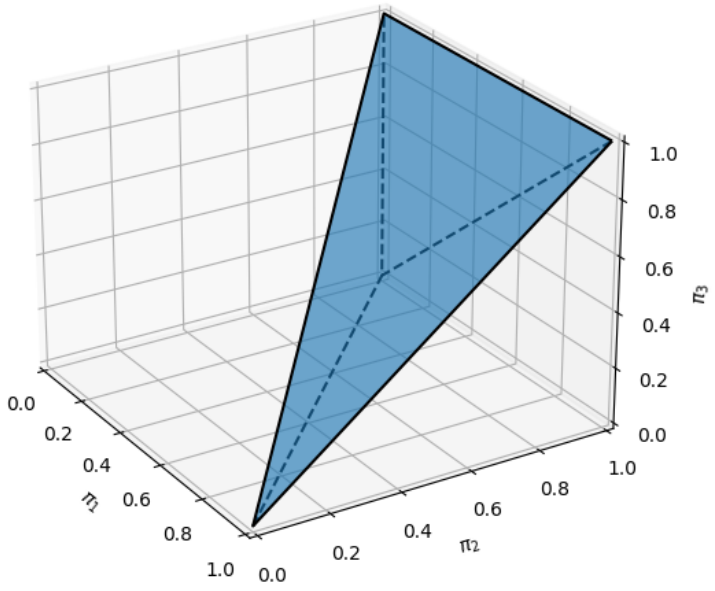

Figure 2.2. Polytope representing consistent truth values for a set of sentences $S_1 = \forall x : P(x)$, $S_2 = \forall x : P(x) \Rightarrow Q(x)$ and $S_3 = \forall x : Q(x)$ (the image is a rotated remake of Fig. 2 in (Nilsson, 1986 [2], p. 76))

As an example we will now consider sentences S_1 and S_2 as the knowledge base and we will calculate the truth value of S_3, i.e. perform probabilistic entailment

$$\{\forall x : P(x),\ \forall x : P(x) \Rightarrow Q(x)\} \vdash \{\forall x : Q(x)\}.$$

In accordance with Figure 2.2 we'll assign some consistent truth values to the formulas in the knowledge base, for example $\pi_1 = \pi(S_1) = 0.6$ and $\pi_2 = \pi(S_2) = 0.7$. Then we can use Equation (2.1) to solve for π_3 as following:

1. Add vectors of 1 as the first row into V and Π. This may be interpreted as adding tautology to the knowledge base, but it is also a way to enforce the constraint $\sum p_i = 1$.

$$\begin{bmatrix} 1 \\ \Pi \end{bmatrix} = \begin{bmatrix} \mathbf{1} \\ V \end{bmatrix} \cdot P \ \Rightarrow\ \begin{bmatrix} 1 \\ 0.6 \\ 0.7 \\ \pi_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}$$

2. Eliminate the last rows of V and Π and calculate P from the modified matrices V', Π'. Generally the equation is under-determined (and this holds in our example) as the number of possible worlds is usually higher than the number of sentences present in the probabilistic entailment, therefore we should expect the solution for P will not be unique.

$$\Pi' = V'P \Rightarrow \begin{bmatrix} 1 \\ 0.6 \\ 0.7 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}$$

Formally we could proceed with multiplying the equation with left pseudo-inverse of V' but in this trivial case we can caluclate P by solving the system of linear equations:

$$\begin{aligned}
0.6 &= p_1 + p_2 & 0.7 &= p_1 + p_3 + p_4 & 1 &= p_1 + p_2 + p_3 + p_4 \\
p_1 &= 0.6 - p_2 & 0.7 &= 0.6 - p_2 + p_3 + p_4 & 1 &= 0.7 - p_2 \\
& & p_2 &= -0.1 + p_3 + p_4 & 0.3 &= -0.1 + p_3 + p_4 \\
p_1 &= 0.3 & p_2 &= 0.3 & p_3 + p_4 &= 0.4
\end{aligned}$$

3. Enforce non-negativity constraint $p_i \geq 0$ on possible values of P and check that P may actually represent a probability distribution — this may not hold if the initial truth values for sentences in knowledge base were assigned inconsistently. In our example the check passes and we find boundaries for p_3 and p_4 as:

$$p_3 \in [0.0, 0.4],\ p_4 \in [0.0, 0.4]\ , p_3 + p_4 = 0.4$$

4. Denote the last row of V (the one eliminated in step 2) as S. Target probability π_3 then may be calculated as:

$$\begin{aligned}
\pi_3 &= SP \\
\pi_3 &= [\,1 \quad 0 \quad 1 \quad 0\,] \cdot [\,0.3 \quad 0.3 \quad p_3 \quad p_4\,]^T \\
\pi_3 &= 0.3 + p_3 \\
\pi_3 &\in [0.3, 0.7]
\end{aligned}$$

As we can see, the result of the probabilistic entailment is not unique, but gives us only possible bounds on the values of π_3. More intuitive picture of the situation is shown in Figure 2.3, where the calculation is described in a geometric way as finding intersection of the polytope of consistent values with planes $\pi_1 = 0.6$ and $\pi_2 = 0.7$.

If we need to select only one solution, we may calculate π_3 from the probability distribution over the possible worlds with the largest entropy, as this is the one about which we know least prior information [4]. Entropy H of probability distribution $\mathbf{p}$ is defined as [5]:

$$H = -\sum p_i \log p_i$$

Maximization of H could be solved using the method of Lagrange multipliers, however in our example where p_1 and p_2 are already set and the only constraint on p_3 and p_4 is $p_3 + p_4 = 0.4$ we may conclude that the maximal entropy will be reached when $p_3 = p_4$, i.e. $p_3 = 0.2$ and $p_4 = 0.2$. The probabilistic truth value of sentence S_3 for this solution is $\pi_3 = 0.3 + 0.2 = 0.5$.

Following list summarizes the facts about Nilsson's probabilistic logic that were described in this section:

- Calculation of probabilistic truth values may be performed in a form of matrix equations, however as the first step all consistent truth values assignments in the possible worlds must be enumareted, and the complexity of the enumeration grows exponentialy in the nubmer of sentences N.
- Assignment of initial probabilistic truth values π_i to the sentences in the knowledge base must be performed carefully because a random assignment may also be inconsistent.
- Even if the initial assignment of π_i is consistent, the probabilistic entailment usually doesn't provide a unique solution to probability of entailed sentences. In this case we may choose the solution associated with the distribution over possible worlds P having the largest entropy.

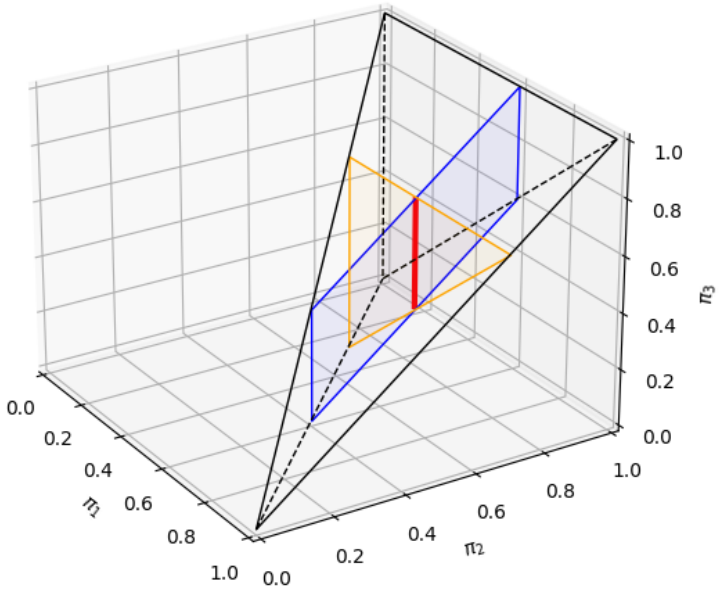

Figure 2.3. Intersection of the polytope from Figure 2.2 with planes $\pi_1 = 0.6$ (blue) and $\pi_2 = 0.7$ (orange). The red segment is the intersection of the planes and the polytope and represents admissible values for π_3 (interval $[0.3, 0.7]$).

2.2 Probabilistic Graphical Models

This section describes two most commonly employed probabilistic statistical models — the first is a *Bayesian network* and the other one is a *Markov random field* (MRF), sometimes called analogically with the first model a *Markov network*. The models were devised as an approach to encode dependency relations between random variables as a graph and then exploiting this knowledge for an efficient evaluation of random fields and their underlying joint probability distributions, also utilizing methods of the graph theory.

The models are based on the *chain rule* for calculation of joint probability distributions of multiple random variables. The chain rule is a generalization of an observation that the joint probability distribution of two random variables X, Y may be expressed as a product of the marginal probability of one variable and the conditional probability of the other given the first one:

$$\mathrm{P}(X, Y) = \mathrm{P}(X \mid Y) \cdot \mathrm{P}(Y)$$

In order to generalize this observation for multiple random variables we only need to apply the rule for one variable at time, always conditioning on the rest of not-yet entered variables, until the last one is reached:

$$\begin{aligned} \mathrm{P}(X_1, X_2, ..., X_n) &= \mathrm{P}(X_1 \mid X_2, ..., X_n) \cdot \mathrm{P}(X_2, ..., X_n) && (2.2) \\ &= \mathrm{P}(X_1 \mid X_2, ..., X_n) \cdot \mathrm{P}(X_2 \mid X_3, ..., X_n) \cdot \mathrm{P}(X_3, ..., X_n) && (2.3) \\ &= ... \\ &= \mathrm{P}(X_1 \mid X_2, ..., X_n) \cdot \mathrm{P}(X_2 \mid X_3, ..., X_n) \cdot ... \cdot \mathrm{P}(X_n) && (2.4) \end{aligned}$$

Actual order of the variables may be of course different as long as the intention of the chain rule is followed. In the thesis we will also use a shorthand notation $p(x_1, x_2, ...x_n)$

for probability of actual assignment of values to random varibles (analogically also for conditional probabilities):

$$p(x_1, x_2, ...x_n) = \mathrm{P}(X_1 = x_1, X_2 = x_2, ..., X_n = x_n)$$

Equation (2.4) is a good insight into splitting the calculation of the full joint probability distribution into the number of more tractable factors which could be represented by smaller probability tables or functions with less variables than the ones for the full joint probability. However applying the chain rule exactly in the form of Equation (2.4) doesn't actually considerably reduce the complexity. If we consider discrete random variables and denote the size of the largest domain of values for any X_i as K, evaluation of the left hand side requires construction of a probability table with $\mathcal{O}(K^N)$ elements, while evaluating first expression on the right hand side requires construction of a conditional probability table for (up to) K possible values of X_1 conditioned on $\mathcal{O}(K^{N-1})$ values for the rest of variables, i.e. the time complexity generally remains the same $\mathcal{O}(K^N)$.

The key problem in evaluating the Equation (2.4) is that each variable is conditioned on all remaining variables, while in practice most of the remaining variables influence the value of the conditional probability only negligible or not at all. This is captured in the concept of *conditional independence* [6].

Definition 2.1. (Conditional independence) *Two random variables A, B are conditionally independent given a random variable C (denoted $A \perp\!\!\!\perp B \mid C$) if and only if they are independent in their conditional probability distribution given C for all possible values of A,B,C*:

$$\mathrm{P}(A, B \mid C) = \mathrm{P}(A \mid C) \cdot \mathrm{P}(B \mid C) \tag{2.5}$$

The defition of conditional independence may be equivalently rephrased as follows — if we're given conditional probability $\mathrm{P}(A \mid C)$ and know $A \perp\!\!\!\perp B$, observing B has no effect on the value of the conditional probability, that is:

$$A \perp\!\!\!\perp B \mid C \;\Leftrightarrow\; \mathrm{P}(A \mid B, C) = \mathrm{P}(A \mid C) \tag{2.6}$$

Conditional independence may be also generalized for sets of random variables — actually it is more or less sufficient just to interpret random variables A, B, C in Definition 2.1 as sets of random variables. Equation (2.6) then may be used to simplify factors of the joint probability distribution if we can efficiently represent conditional (in)dependencies between variables, because as the equation suggests, all conditionally independent variables then may be ignored and the conditional probability tables may be calculated only w.r.t. conditioning variables. As an example, we may simplify calculation of the probability of X_1 in Equation (2.4) if we know that X_1 is conditionally independent on all other variables given X_2, X_5 as

$$\begin{aligned}\mathrm{P}(X_1, X_2, ..., X_n) &= \mathrm{P}(X_1 \mid X_2, ..., X_n) \cdot \mathrm{P}(X_2 \mid X_3, ..., X_n) \cdot ... \cdot \mathrm{P}(X_n) \\ &= \mathrm{P}(X_1 \mid X_2, X_5) \cdot \mathrm{P}(X_2 \mid X_3, ..., X_n) \cdot ... \cdot \mathrm{P}(X_n)\end{aligned}$$

The process then may be similarly repeated for conditional probability of X_2 and another random variables present in the equation.

As a last point before proceeding to the description of two most common probabilistic graphical models — Bayesian networks and Markov networks — we should note that conditional independence of random variables is not related to their standard independence. Two random variables may be independent on each other but conditionally dependent given another variable, and vice versa.

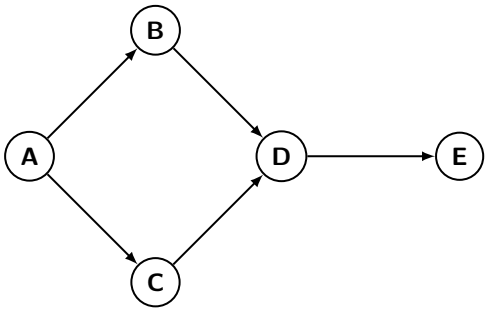

Figure 2.4. Graph of Bayesian network of 5 variables.

For example of two independent variables that become conditionally dependent let's consider rolling two fair six-sided dice, denote the result of the first die A and the result of the other B. As usually in such a case we expect that results of each roll are independent so $\mathrm{P}(A, B) = \mathrm{P}(A) \cdot \mathrm{P}(B)$. However, when we also observe variable C which checks if sum of rolls is even or odd, A and B become conditionally dependent given C — knowing that the sum of rolls is even doesn't provide any additional information without also knowing the result of the other die, so the conditional probability is equal to the marginal (same applies to $\mathrm{P}(B \mid C)$):

$$\mathrm{P}(A = a \mid C = c) = \mathrm{P}(A = a) = \frac{1}{6}.$$

However if we know that $C = even$ and $A = 3$, then we see that B must be also odd, so if we take even value $B = 2$, Equation (2.5) doesn't hold and therefore $A \not\perp\!\!\!\perp B \mid C$:

$$\mathrm{P}(A = 3 \mid C = even) \cdot \mathrm{P}(B = 2 \mid C = even) = \frac{1}{6} \cdot \frac{1}{6} = \frac{1}{36} \neq$$
$$\neq \mathrm{P}(A = 3, B = 2, C = even) = 0$$

2.2.1 Bayesian Networks

Bayesian network is a directed acyclic graph (DAG) where vertices represent variables of interest (random variables, parameter models, hypotheses) and oriented edges represent conditional dependencies between the variables; oriented edge $X_u \to X_v$ specifies that X_v is conditionally dependent on X_u. Edge direction however primarily captures the real causal connections and not the actual direction used for computations, because the information necessary for reasoning can still be propagated in both ways [7].

The most important property of Bayesian networks is that every vertex X is independent from its non-descendants given set of its parent vertices Pa_X. Computation of the marginal probability of variable X is then conditioned on the parent nodes and only requires knowledge of their probabilities:

$$\mathrm{P}(X) = \mathrm{P}(X \mid Pa_X) \tag{2.7}$$

Probabilities of parent nodes are usually stored in the child node in a form of conditional probability table. Provided the number of parents for each node is bounded, the number of required conditional distributions for each node grows only linearly in the size of the Bayesian net, which is a considerable improvement over exponential growth for Equation (2.2).

Computation of full joint probability distribution in the Bayesian net is factorized into product of conditional distributions conditioned on parent nodes:

$$\mathrm{P}(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} \mathrm{P}(X_i \mid Pa_{X_i})$$

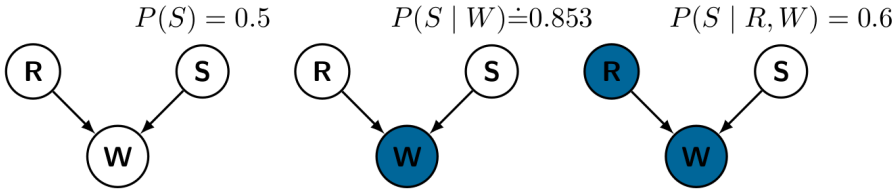

Figure 2.5. Illustration of Bayesian net described in the calcualtion example. **R** represents raining, **S** sprinkler and **W** a wet pavement. Initial situation is captured in the left graph, in the central graph we observe the pavement is wet which influences marginal probabilities of both **R** and **S**. In the right graph we find out that it was actually raining, but this information also affects our knowledge about **S**, because they become dependent after observing **W**.

Let's take as an example the Bayesian network presented in Figure 2.4. The joint probability distribution of the network may be expressed as:

$$\mathrm{P}(A, B, C, D, E) = \mathrm{P}(A) \cdot \mathrm{P}(B \mid A) \cdot \mathrm{P}(C \mid A) \cdot \mathrm{P}(D \mid B, C) \cdot \mathrm{P}(E \mid D)$$

More illustrative example which will also point to a not so obvious property of Bayesian networks is illustrated in Figure 2.5. In the morning, we may observe that the pavement in front of the house is wet. There are two possible causes for this — it may have been raining during the night or early in the morning the sprinkler on the grass was on. The sprinkler should be watering the grass every morning, but it is faulty and works more or less randomly. It also doesn't have any detector to check whether the grass is already wet, so it may also turn on even if it was raining. We have these prior probabilities for the sprinkler (S) and the raining (R):

$$\mathrm{P}(S = on) = 0.5$$
$$\mathrm{P}(R = true) = 0.2$$

The conditional probabilities for observing wet pavement (W) given the other two events are stated as follows:

$$\mathrm{P}(W = wet \mid S = on, R = true) = 0.9$$
$$\mathrm{P}(W = wet \mid S = on, R = false) = 0.7$$
$$\mathrm{P}(W = wet \mid S = off, R = true) = 0.6$$
$$\mathrm{P}(W = wet \mid S = off, R = false) = 0.01$$

Now in the morning we actually observe the pavement is wet and we may want to evaluate the posterior probability that the sprinkler was on. This may be done using Bayes' theorem:

$$\mathrm{P}(S \mid W) = \frac{\mathrm{P}(W \mid S) \cdot \mathrm{P}(S)}{\mathrm{P}(W)} \tag{2.8}$$

The denominator is evaluated by marginalizing over R, S:

$$\begin{aligned}\mathrm{P}(W = wet) &= \sum_{s \in \{on, off\}} \sum_{r \in \{true, false\}} \mathrm{P}(W = wet \mid S = s, R = r) \cdot \mathrm{P}(S = s) \cdot \mathrm{P}(R = r) \\ &= 0.434\end{aligned}$$

Similarly for conditional probability $\mathrm{P}(W \mid S)$:

$$\mathrm{P}(W = wet \mid S = on) = \sum_{r \in \{true, false\}} \mathrm{P}(W = wet \mid S = on, R = r) \cdot \mathrm{P}(R = r) = 0.74$$

So plugging all the numbers into Equation (2.8) we get:

$$\mathrm{P}(S = on \mid W = wet) = \frac{0.74 \cdot 0.5}{0.434} \doteq 0.853$$

We see that $\mathrm{P}(S = on \mid W = wet) > \mathrm{P}(S = on)$ so observing that the pavement is wet makes it more likely that it the sprinkler was on, which is something we would intuitively expect. Now let's see if something changes when we find out that it was raining in the night (e.g. from a weather report). The posterior probability for the sprinkler changes to:

$$\mathrm{P}(S \mid W, R) = \frac{\mathrm{P}(W \mid S, R) \cdot \mathrm{P}(S) \cdot \mathrm{P}(R)}{\sum_{s \in S} \mathrm{P}(W \mid S, R) \cdot \mathrm{P}(S) \cdot \mathrm{P}(R)}$$

$$\mathrm{P}(S = on \mid W = wet, R = true) = \frac{\mathrm{P}(W = wet \mid S = on, R = true) \cdot \mathrm{P}(S = on)}{\sum_{s} \mathrm{P}(W = wet \mid S = s, R = true) \cdot \mathrm{P}(S = s)} = 0.6$$

After observing that it was raining the probibility that sprinkler was on drops, even though initially these two variables were independent. They however became coupled when we observed the actual value of their common child.

As we can see from the previous example, even though the Bayesian network is a directed graphical model, the information may still flow in any direction when reasoning and evidence provided in the descendant node actually influenced the marginal probability of the parent node. Earlier in the beginning of the section it was declared that a node is conditionally independent from its non-descendants given its parents. This is indeed true, but we may be actually also interested in which nodes actually separate the node from the rest of the network, so we know which nodes may influence reasoning about the node and which are irrelevant.

Identification of separating set of nodes may be defined in terms of *d-separation*, which is based on a notion of *active paths*. First we should consider what configuration of nodes w.r.t. directed edges may be observed over triplets of nodes [8]:

1. *Cascade*: $A \to B \to C$ or $A \leftarrow B \leftarrow C$
   If B is observed, then $A \perp\!\!\!\perp C \mid B$, because we can determine output of C solely on B and A doesn't influence it. If B is unobserved, then $A \not\perp\!\!\!\perp C$, because observing A provides information about B and in turn we may also reason about C.
2. *Common parent*: $A \leftarrow B \to C$
   Reasoning is actually the same as above — if B is observed, $A \perp\!\!\!\perp C \mid B$, otherwise $A \not\perp\!\!\!\perp C$.
3. *V-structure*: $A \to B \leftarrow C$
   The results in this case are opposite to previous ones — if the common descendant B is *unobserved*, then parents are independent — $A \perp\!\!\!\perp C$. But when B is observed, then $A \not\perp\!\!\!\perp C \mid B$. This is also called *explaining away*.

These checks may be recursively applied on larger sets of variables in the graph, leading to a notion of *active paths* in Bayesian network. An undirected path in the Bayesian network is active given a set of observed variables O if for every consecutive triple of variables X, Y, Z one of the following holds:

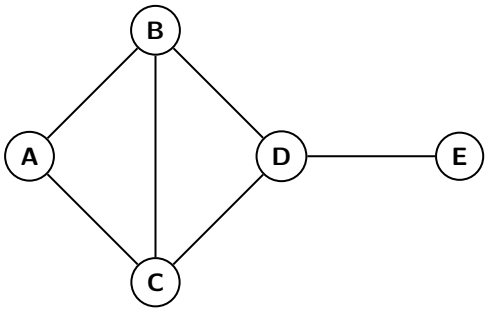

Figure 2.6. Graph of Markov random field of 5 variables with two 3-cliques $\{A, B, C\}$ and $\{B, C, D\}$ and one 2-clique $\{D, E\}$.

- $X \to Y \to Z$ and Y is unobserved ($Y \notin O$),
- $X \leftarrow Y \leftarrow Z$ and Y is unobserved,
- $X \leftarrow Y \to Z$ and Y is unobserved,
- $X \to Y \leftarrow Z$ and Y is or *any of its descendants* is observed.

The independence of sets in Bayesian networks is then specified using *d-separation*. Two sets of variables A, B are d-separated given set O if there is no active path connecting A and B given O. Then set O is also a separating set of sets A, B. Separating set is not actually unique — adding a variable which is not in A or B into the separating set still yields a separating set. The minimal separating set is a separating set from which no variable can be removed without violating d-separation property. In Bayesian networks, the minimal separating set for a variable from the rest of graph consists of variable's parents, its immediate children and all other parents of these immediate children.

2.2.2 Markov Random Fields

Markov random field (MRF) or *Markov network* is a graphical probabilistic model that represents dependencies between variables as an undirected graph. An MRF may be also cyclic, therefore it may, unlike Bayesian networks, conveniently represent cyclic dependencies. Also the notion of separating set for a node is simpler in MRFs as it consists only from all neighbours of the node in question [9].

If graph $G = (V, E)$ represents an MRF, it must satisfy following three Markov properties, ordered from the weakest to the strongest (variable represented by vertex v is denoted as X_v) [10]:

1. **Pairwise Markov property**:
   Any two non-adjacent variables are conditionally independent given all other variables:
   $$X_v \perp\!\!\!\perp X_u \mid X_{V \setminus \{u,v\}}$$
2. **Local Markov property**:
   A variable is conditionally independent of all other variables given its neighbors:
   $$X_v \perp\!\!\!\perp X_{V \setminus N[v]} \mid X_{V \setminus N(v)}$$
   where $N(v)$ is the set of neighbors of v and $N[v] = v \cup N(v)$ is the closed neighbourhood of v.
3. **Global Markov property**:
   Any two subsets of variables are conditionally independent given a separating subset:
   $$X_A \perp\!\!\!\perp X_B \mid X_S$$

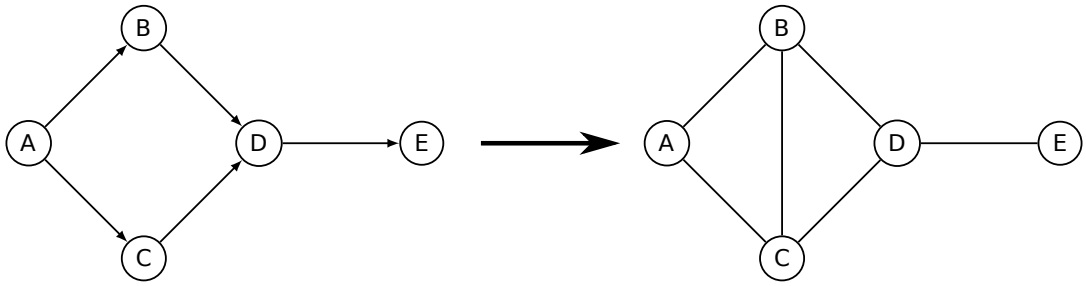

Figure 2.7. Moralization of a Bayesian network (left) into a Markov random field (right).

where X_A, X_B are sets of vertices and X_S is their separating subset (i.e. all paths between a node from X_A to a node in X_B pass through a node in X_S).

All three Markov properties are actually equivalent if the underlying probability distribution induced by variables in the graph is strictly positive.

Computation of the full joint probability distribution in MRFs can be factorized similarly to Bayesian networks as a product of quantities over sets of variables. Unlike the Bayesian networks the quantity is not represented in a form of probability tables, but as a *potential function*. The factorization is then performed over maximal cliques of a graph (graph clique is a fully-connected subgraph of the graph[1]):

$$p(x_1, x_2, ..., x_n) = \frac{1}{Z} \prod_{C \in cl(G)} \phi(C),$$

where $cl(G)$ is the set of maximal cliques of graph G, $\phi(C)$ is a potential function associated with assignments to all variables (vertices) in clique C, and Z is the partition function. This function ensures that the result is actually a probability distribution by summing potential functions for all possible configurations of MRF:

$$Z = \sum_{x_1, x_2 ... x_n} \prod_{C \in cl(G)} \phi(C)$$

As an actual example we present factorization of MRF from Figure 2.6, the set of maximal cliques is $cl(G) = \{\{A, B, C\}, \{B, C, D\}, \{D, E\}\}$ (note that if there was an edge connecting A, D the 3-cliques would be replaced with a 4-clique $\{A, B, C, D\}$) and the probability of the configuration factorizes into:

$$p(a, b, c, d, e) = \frac{1}{Z} \cdot \phi(a, b, c) \cdot \phi(b, c, d) \cdot \phi(d, e)$$

Two problems however arise when we try to perform exact inference in MRFs. The first one is that listing all maximal cliques in the graph is NP-complete (problem of detection of any clique of size k is actually listed in *Karp's 21 NP-complete problems*[11]). This may be resolved due to the fact that in practice structures of MRFs are usually not random, but they are designed intentionally, so the structure of maximal cliques is known in advance and it is therefore unnecessary to detect them algorithmically. The second problem is the evalution of the partition function which requires summation over all possible assignments, which is in general NP-hard. This problem may not be resolved as easily as the first one and exact inference in MRFs remains intractable in general, even though in some MRF classes this calculation may be performed efficiently.

[1] We may prepend a clique with a number of vertices present in it, i.e. 3-clique, 4-clique etc. 2-clique is an edge and 1-clique is just a vertex

There are also procedures transforming Bayesian network into MRF and vice versa. As a first step in transformation of Bayesian network into MRF we only need to trivially substitute every directed edge with an undirected one. As a second step we need to add an edge between all vertices, which share a direct descendant and are disconnected in the Bayesian network. This is called *moralization* as it enforces a relation between parent nodes (a "marriage", though it may easily result in a polygamy if the node has more than 2 parents). If the second step is omitted, we lose information that the value of the child node is actually dependent on values of all its parents simultaneously. The procedure is illustrated in Figure 2.7. Potential functions for each clique then correspond with joint probability of all variables in the clique, which may be in turn computed from the conditional probability table associated with the leaf node of the clique by Equation (2.7). The partition function of such a transformed net is trivially 1 (all probabilities in Bayesian networks must sum to 1). The converse process of transforming an MRF into a Bayesian net is called *triangulation*, but it is rarely used, because it is usually intractable (it often results in an almost fully connected DAG).

Chapter 3
Markov Logic Networks

This chapter describes *Markov logic networks* (MLN), a probabilistic logic framework used in the statistical relational learning (SRL). Markov logic networks encode statistical regularities in a from of weighted logical formulas. The following section provides definitions of MLNs and related concepts, then their basic properties, means of inference and standard learning tasks in MLNs are discussed. Finally we'll focus on the key concept of the thesis — *relational marginal polytope* which originates from relational marginal problem — a task concerned with finding the maximum-entropy probability distribution satisfying specified marginal probabilities.

3.1 Definition

The concept of Markov logic networks first appeared in the paper of Richardson and Domingos in 2006 [12]. The rationale behind their proposal is that when we model a problem using first-order logic formulas (these form a knowledge base), the formulas are actually hard-constraints and any potential world that violates just one of them is consequently impossible. This behaviour however may not be always desirable as often a formula that doesn't hold in all cases may still capture useful information about modelled relationships. In order to soften the constraint checking a weight is associated with each formula. The weight should represent how important the constraint is in the model — the higher the weight, the higher the importance of the constraint. In this setting the world violating a constraint doesn't become instantly impossible, only less probable. If the world violates higher number of constraints or if it violates more important ones, the world's probability decreases proportionally.

Definition 3.1. (Markov logic network): *A Markov logic network (MLN) is a set of weighted first-order logic formulas* (α, w) *where* $w \in \mathbb{R}$ *and* α *is function-free and quantifier-free first-order logic formula.*

MLN Φ induces a probability distribution over a set of possible worlds Ω:

$$\text{for } \omega \in \Omega : \; p_\Phi(\omega) = \frac{1}{Z} \exp\left(\sum_{(\alpha,w)\in\Phi} w \cdot N(\alpha,\omega) \right) \tag{3.1}$$

In this equation $p_\Phi(\omega)$ denotes probability of observing possible world ω, $N(\alpha,\omega)$ is total number of groundings of formula α that are satisfied in ω relative to a finite set of constants Δ (called the domain) and Z is the *partition function* that normalizes the result so it forms a probability distribution similarly as in MRFs. Presence of the normalizing term Z draws exact inference in MLNs generally intractable in the same way as in MRFs, as its evaluation requires summation over all possible worlds and the count of these possible worlds is exponential in the size of the domain.

An MLN can be created from a first-order logic knowledge base just by assigning arbitrary weights to each formula in the KB. The first-order logic is actually a special case of MLN where all weights are infinite, i.e. any violation of a formula renders

the associated world impossible. The probability distribution over satisfiable possible worlds in this case is uniform. The weight of the formula can be interpreted as a log-odd between observing a world where the formula holds and a world where it doesn't, assuming all remaining weights are equal.

3.1.1 Relation to MRF

Markov logic networks are closely related to Markov random fields — grounding an MLN with respect to a domain creates an instance of a MRF and in this sense MLNs may be considered templates for a variety of MRFs. The resulting MRFs may vary significantly in size but they will share common structures. The procedure for grounding MLN into MRF was described in the initial paper by Richardson and Domingos [12]). An instance of MRF $M_{\Phi,\Delta}$ may be created from MLN Φ with respect to the domain Δ in following steps:

1. Create one binary node in $M_{\Phi,\Delta}$ for each possible grounding of each predicate appearing in MLN Φ. The value of the node is 1 if the ground atom is true, and 0 otherwise.
2. For each possible grounding of each formula α_i in MLN Φ create one feature in $M_{\Phi,\Delta}$. The value of this feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is weight w_i associated with formula α_i in MLN Φ.

3.2 Inference

Exact inference in MLNs is in general intractable for similar reasons as in MRFs — the partition function Z is calculated as a sum of terms over all possible worlds, and the number of all possible worlds in general grows exponentially w.r.t the size of domain $|\Delta|$.

3.2.1 Weighted Model Counting

Calculation of the partition function may be converted to the *weighted first-order model count* problem (WFOMC)[13]:

Definition 3.2. (WFOMC): *Let $w(P)$ and $\overline{w}(P)$ be functions from predicates to real numbers (w and $\overline{w}$ are called weight functions) and let Φ be a first-order theory. Then*

$$\mathrm{WFOMC}(\Phi, w, \overline{w}) = \sum_{\omega \in \Omega : \omega \models \Phi} \prod_{a \in \mathcal{P}(\omega)} w(Pred(a)) \prod_{a \in \mathcal{N}(\omega)} \overline{w}(Pred(a)) \quad (3.2)$$

where $\mathcal{P}(\omega)$ and $\mathcal{N}(\omega)$ denote the positive literals that are true and false in ω, respectively, and $Pred(a)$ denotes the predicate of a (e.g. $Pred(friends(Alice, Bob)) = friends$).

The evaluation of WFOMC then proceeds with addition of a formula ξ_i for every weighted formula (α_i, w_i) in Φ whose free variables are exactly $x_1, x_2, ...x_k$:

$$\forall x_1, ..., x_k : \xi_i(x_1, ..., x_k) \Leftrightarrow \alpha_i(x_1, ..., x_k)$$

Then we set $w(\xi_i) = \exp(w_i)$, $\overline{w}(\xi_i) = 1$ for all new predicates and $w(\alpha_i) = 1$ and $\overline{w}(\alpha_i) = 1$ for the original predicates. If we denote the resulting set of predicates Γ, it will turn out that actually $\mathrm{WFOMC}(\Gamma, w, \overline{w}) = Z$. WFOMC may be also used for evaluation of the marginal probability of query q under Γ:

$$\mathrm{P}_{\Phi,\Omega}(q) = \frac{\mathrm{WFOMC}(\Gamma \cup \{q\}, w, \overline{w})}{\mathrm{WFOMC}(\Gamma, w, \overline{w})}$$

The WFOMC however doesn't change asymptotical complexity of computation of the partition function w.r.t. the domain (it remains exponential). However there are classes of MLNs where inference may be performed more efficiently, in polynomial time w.r.t. to the size of the domain. These problems are called *domain liftable*.

Definition 3.3. (Domain liftability) *An algorithm for computing* WFOMC *is said to be domain-liftable if it runs in time polynomial in the size of the domain.*

Example of domain-liftable MLN instances are MLNs where each predicate contains at most two variables [14].

WFOMC is a generalization of *weighted model counting* (WMC), which is in turn generalization of model count for propositional formulas. The model counting task simply counts number of distinct satisfying assignments to boolean variables in the formula, WMC extends the task by also associating weights to variables. Distinct weights might be associated in case when the boolean value of the variable is true and when it is false, formula for WMC$(F, w, \overline{w})$ is then analogical to Equation (3.2) [15]:

$$WMC(F, w, \overline{w}) = \sum_{\theta:\theta(F)=1} \left(\prod_{i:\theta(X_i)=1} w(X_i) \cdot \prod_{i:\theta(X_i)=0} \overline{w}(X_i) \right)$$

(F is propositional formula with variables X_i, $w(X_i)$ and $\overline{w}(X_i)$ are weight functions for positive, resp. negative occurence of the variable and θ is an assignment of variables to $\{0, 1\}$).

WFOMC for formula Φ is then defined as WMC over $F_{\Phi,n}$ — the propositional grounding of the formula Φ over domain n — and the weight functions $w, \overline{w}$: Tup(n) $\rightarrow \mathbb{R}$ (Tup(n) denotes set of all possible groundigs of Φ over n) and WFOMC is defined as WFOMC(Φ,n,w,$\overline{w}$) = WMC($F_{\Phi,n}$,w,$\overline{w}$).

WFOMC as defined in this section is so called *symmetric WFOMC*, as the weight for every element of Tup(n) depends only on the name of the predicate, so for example all groundings of predicate *edge(X, Y)* share the same weights w, resp. $\overline{w}$ (but w and $\overline{w}$ may be different). This definition is generally used in the literature related to MLNs. Just as a side note we remark that there is also class of *asymmetric WFOMC* where the weights may depend even on the actual domain elements present in the grounding of the predicate, e.g. it may hold that $w(foo(John, Mary)) \neq w(foo(Peter, Kate))$.

3.3 Relational marginal polytopes

This section introduces *relational marginal polytopes* (RMP) with which calculation the thesis is mainly concerned. RMPs emerge as a set of feasible solutions to the *relational marginal problems* which try to find weights for a maximum-entropy distribution over the possible worlds w.r.t. statistical marginal probabilities of formulas in the MLN.

3.3.1 Polytopes

Polytope is a geometric object generalizing three-dimensional polyhedrons into arbitrary number of dimensions. Polytope with dimension n is denoted as n-polytope and its sides consist of (arbitrary number of) $(n-1)$-polytopes that may share a common $(n-2)$-polytope (and so on). Standard terminology for elements of n-polytope with specific dimensions are *vertex* (0-dimensional — point), *edge* (1-dimension) and *facet* ($(n-1)$-dimension).

The thesis considers only *bounded convex polytopes*. There are two common representations (and also definitions) of convex polytope:

- **V-representation** (vertex) — bounded convex polytope is defined as the convex hull of a finite set of points. The minimal V-representation of the polytope is given by the set of its vertices.
- **H-representation** (half-space) — bounded convex polytope is defined as an intersection of a finite number of half-spaces. Half-space may be written as a linear inequality:

$$a_1x_1 + a_2x_2 + ... + a_nx_n \leq b$$

The minimal H-representation of the polytope consists of the set of inequalities defining its facets.

3.3.2 Relational marginal problem

The total number of satisfiable formula groundings $N(\alpha, \omega)$ in Equation (3.1) presents the absolute number of admissible groundings. It may be however more convenient to express this quantity relatively to the size of the number of possible groundings. This quantity is called *formula statistic* w.r.t. the possible world ω:

Definition 3.4. (Formula statistic) *Let α be a quantifier-free first-order logic formula with k variables $\{x_1, ...x_k\}$. Its formula statistic w.r.t. a possible world ω is defined as*:

$$Q_\omega(\alpha) = \left(\frac{|\Delta|}{k}\right)^{-1} \cdot (k!)^{-1} \cdot N(\alpha, \omega) \tag{3.3}$$

There $|\Delta|$ denotes size of the domain and k denotes arity of predicate α. Intuitively the formula statistic represents the probability that a random injective substitution of variables that ground formula α will be satisfied in the possible world ω if we draw the substitution randomly from the uniform distribution.

With notion of formula statistics, we may continue to the definition of the *relational marginal problem*.

Definition 3.5. (Relational marginal problem): *The relational marginal problem is a convex optimization task with the following formulation*:

$$\min \sum_{P_\omega : \omega \in \Omega} P_\omega \log P_\omega \quad \text{s.t.} \tag{3.4}$$

$$\forall i : 1, ..., l : \sum_{\omega \in \Omega} P_\omega \cdot Q_\omega(\alpha_i) = \theta_i \tag{3.5}$$

$$\forall \omega \in \Omega : P_\omega \geq 0, \sum_{\omega \in \Omega} P_\omega = 1 \tag{3.6}$$

where P_ω denotes the probability of possible world ω, $Q_\omega(\alpha_i)$ is formula statistic associated with formula α_i in the particular possible world, and θ_1, ... θ_k are the target expected values for each formula statistics, also called the *relational marginals* (hence the name of the task).

To provide a more thorough analysis of the formulation — Equation (3.4) minimizes *negative* entropy of the probability distribution over the possible worlds, Equation (3.5) represents constraints specified by the relational marginals and the last Equation (3.6) ensures the result of the task is a probability distribution. Assuming there exists a strictly positive feasible solution to the task, the optimal solution is an MLN

$$P_\omega = p_\Phi(\omega) = \frac{1}{Z} \exp\left(\sum_{(\alpha_i, \lambda_i) \in \Phi} \lambda_i \cdot Q_\omega(\alpha)\right)$$

where λ_i are obtained by maximizing dual criterion which is incidentally MLN's log-likelihood w.r.t. some training example with statistics for each formula equal to expected ones:

$$L(\lambda) = \sum_{\alpha_i} \lambda_i \cdot \theta_i - \log \sum_{\omega \in \Omega} e^{\sum_{\alpha_i} \lambda_i \cdot Q_\omega(\alpha_i)} \tag{3.7}$$

Note that the second term in Equation (3.7) is a logarithm of the partition function Z for an MLN with formula weights λ. Due to the duality if we are able to efficiently solve relational marginal problems, we can also efficiently solve maximum likelihood estimation of MLN. For optimization of $L(\lambda)$ using gradient-based methods we also need to calculate partial derivates:

$$\frac{\partial L}{\partial \lambda_i} = \theta_i - \frac{\sum_{\omega \in \Omega} Q_\omega(\alpha_i) \cdot e^{\sum_{\alpha_i} \lambda_i \cdot Q_\omega(\alpha_i)}}{\sum_{\omega \in \Omega} e^{\sum_{\alpha_j} \lambda_j \cdot Q_\omega(\alpha_j)}} \tag{3.8}$$

The denominator of the second term is again the partition function Z, while each term in the numerator consists of a formula statistic of the formula α_i associated with the weight λ_i in the possible world and an exponential which actually represents the possible world's potential. As the ratio of the world's potential and the partition function is equal to the actual probability of the possible world, second term as a whole represents the expected value of the formula α_i statistics over all possible worlds, i.e. we can also write

$$\frac{\partial L}{\partial \lambda_i} = \theta_i - \mathbb{E}\left[Q_\omega(\alpha_i)\right]$$

Evaluation of both $L(\lambda)$ and the gradient therefore involves computation of the partition function Z. This may be translated to WFOMC(Φ, w, $\overline{w}$) of MLN Φ corresponding to current weights assignment using the procedure described in Section 3.2.1. Computation of the numerator in Equation (3.8) then proceeds similarly using WFOMC($\Phi \cup \{\alpha_i \vartheta\}$, w, $\overline{w}$) where $\alpha_i \vartheta$ is an injective grounding substitution of α_i. Solving the relational marginal problem in general is therefore as hard as evaluation of the partition function, which is #P-hard problem, but as was mentioned earlier, polynomial time algorithms exist for special cases.

3.3.3 RMP Definitions

During solution of relational marginal problems it is possible to encounter a relational marginals that define expected values of formula statistics which are actually not realizable on the domain of the specified size (or on a domain of any size at all). Consider following example, which describes edges and triangles present in a graph in terms of propositional logic [14]:

Example 3.6. : *Consider an MLN Φ consisting of following formulas (weights omitted)*:

- $\phi : edge(x_1, x_2)$,
- $\psi : edge(x_1, x_2) \wedge edge(x_2, x_3) \wedge (x_1, x_3)$.
- $\Delta = \{c_1, c_2 ... c_{100}\}$

Now when considering expected values of formula statistics $\mathbb{E}[(Q_\omega(\phi))] = 0$ and $\mathbb{E}[(Q_\omega(\psi))] = 0.5$, we can easily see that no possible world can conform to this distribution as there simply cannot be even one triangle in a graph without edges. Values of statistics corresponding to some actual probability distributions form so called *relational marginal polytope* [1]:

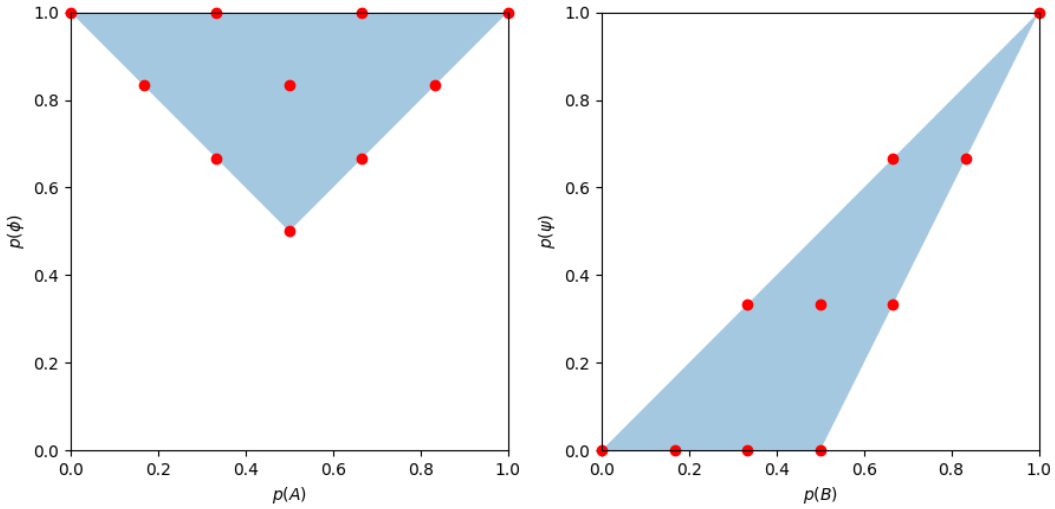

Figure 3.1. Examples of RMP w.r.t. domain of size 3 for two MLNs (both under unique names assumption). Blue area represents RMP, red points denote actual formula statistics Q that can be achieved in the MLN. **Left** $A = a(X,Y)$, $\phi = a(X,Y) \vee \neg\, a(Y,X)$. **Right** $B = b(X,Y)$, $\psi = b(X,Y) \wedge b(Y,X)$.

Definition 3.7. (Relational marginal polytope): *Let Ω be the set of possible worlds on domain Δ and $\Phi = (\alpha_1, ..., \alpha_m)$ be a list of formulas. The relational marginal polytope* $\mathbf{RMP}(\Phi, \Delta)$ *w.r.t. Φ is defined as*:

$$\mathbf{RMP}(\Phi,\Delta) = \{(x_1,...,x_m) \mid \exists \text{ distribution on } \Omega \text{ s.t. } \mathbb{E}[Q(\alpha_1,\omega)] = x_1 \wedge ... \wedge \mathbb{E}[Q(\alpha_m,\omega)] = x_m\}.$$

Relational marginal polytope w.r.t. list of formulas $(\alpha_1, ..., \alpha_l)$ forms a convex hull of a set:

$$\{(Q_\omega(\alpha_1), ..., Q_\omega(\alpha_l)) \mid \omega \in \Omega\}.$$

Important property of RMPs is that RMPs associated with larger domains are *subsets* of RMPs associated with domains which contain less elements. Furthermore, using a notion of *η-interiority* a bound can be provided on the maximal difference between any points in these polytopes.

Definition 3.8. (η-interiority [1]): *Let $\eta > 0$, $\mathbf{P}$ be a polytope and $A^{=}\mathbf{x} = \mathbf{c}$ be the maximal linearly independent system of linear equations that hold for the vertices of $\mathbf{P}$. A point θ is said to be in the η-interior of $\mathbf{P}$ if $\{\theta' | A^{=}\theta' = \mathbf{c}, \|\theta' - \theta\|_2 \leq \eta\} \subseteq \mathbf{P}$.*

Equivalently point y is contained in η-interior of RMP $\mathbf{P}$ if a ball with radius η with center y is subset of the polytope. Regardless on the definition we use, detecting whether a point is in η-interior of RMP is NP problem.

We may also analogically define an integral counterpart to RMP, the *integer relational marginal polytope* (IRMP), which is a convex hull of all realizable grounding counts:

$$\mathbf{IRMP}(\Phi,\Delta) = \{N(\alpha_1,\omega), ..., N(\alpha_m,\omega) : \omega \in \Omega\} \tag{3.9}$$

Both types of marginal polytopes are interchangeable as they are equivalent up to scaling — as mentioned in previous sections, there is a straightforward mapping between number of groundings $N(\alpha_i,\omega)$ and the formula statistics $Q(\alpha_i,\omega)$:

$$Q(\alpha_i,\omega) = |\Delta|^{-|vars(\alpha_i)|} \cdot N(\alpha_i,\omega)$$

Due to this relation, we may switch between RMP and IRMP almost freely and choose the formulation which is more convenient for particular task.

3.3.4 Lifted reduction algorithm

This section describes an algorithm for calculation of IRMP for domain-liftable MLN based on reduction of the task to calculation of partition functions of (other) MLNs devised by Kuželka and Wang [1]. The algorithm runs in polynomial time as calculation of partition functions in auxiliary MLNs is also domain-liftable. The algorithm uses the hyperplane-representation (H-representation) where the polytope is represented as an intersection of half-spaces corresponding to a set of linear inequalities of the form $\vec{a} \cdot \vec{x} \leq b$. Each facet of the polytope corresponds to one of the inequalities.

Before the actual algorithm description we should also note that the maximal size of the resulting IRMP is polynomially bounded by the maximal number of formula groundings. Let i-th coordinate of vector $\vec{x}$ correspond to formula α_i and let $r_i = \max N(\alpha_i, \omega)$ denote the maximal number of groundings of α_i among all possible worlds. The value of x_i can take only values in $\{0, 1, ..., r_i\}$ and since value of each r_i is polynomially bounded in the size of the domain $|\Delta|$ ($r_i \in O(|\Delta|^{|vars(\alpha_i)|})$), so is the maximal size of IRMP. The algorithm proceeds in the following steps:

1. Enumerate all possible normal vectors $\vec{a}$ for hyperplanes that could be present in the H-representation by iterating over all linearly independent m-tuples of points in the bounding polytope. A vector $\vec{v}$ perpendicular to the set of points may be computed for example from their generalized cross product. Both $\vec{v}$ and $-\vec{v}$ must be taken into account so the enumeration of possible normal vectors is exhaustive.
2. Construct a new MLN for each normal vector $\vec{a}$. The MLN contains same predicates P_i as the MLN in question but with modified weights $\{(\alpha_i,\ 2 \cdot a \cdot \ln|\Omega|) :\ 1 \leq i \leq m\}$.
3. Calculate value of the partition function Z of the new MLN. The value of b for the hyperplane is then obtained in this way: if for every possible world ω holds $\sum_i a_i \cdot N(\alpha_i, \omega) \leq b$, then $Z \leq |\Omega|^{2b+1}$. If the latter inequality doesn't hold, then the partition function is greater than $|\Omega|^{2b+2}$. Select b as the smallest integer such that there doesn't exist any possible world for which $\sum_i a_i \cdot N(\alpha_i, \omega) > b$.
4. The previous steps yield a set of linear inequalities describing the IRMP. The set may be further reduced to minimal subset of constraints, and this step may be performed in polynomial time w.r.t the size of the domain.

The number of points in the IRMP is polynomially bounded w.r.t. $|\Delta|$ and so is the number of m-tuples and normal vectors generated in step 1. Calculation of the partition function in step 3 is domain-lifted too , so the whole algorithm is also polynomial in $|\Delta|$, yet we can easily see that the number of tuples in step 1 will be enormous. Therefore we should not expect that the algorithm as stated will be efficient in practice.

3.3.5 RMP role in MLN weight learning

Knowledge of (I)RMP of MLN Φ is beneficiary for solution of maximum-likelihood MLN weight learning task. As was already discussed in Section 3.3.2, the dual to the relational marginal problem involves optimization of function $L(\lambda)$ which happens to be equivalent to log-likelihood of the MLN. The article of Kuželka, Kungurtsev [14] further describes how to solve the weight learning problem and also proves that the task is domain-liftable for MLNs over two-variable fragment of the first-order logic.

The article identifies three important steps towards solution of the task:

1. we need to be able to evaluate the function $L(\lambda)$ and its gradient,
2. we need to efficiently calculate the (I)RMP,
3. we need to establish a method for optimization of weights.

For point 2 we may employ the algorithm described in the previous section or its more efficient alternatives described later in the thesis.

Point 1 — evaluation of likelihood function and its gradient — was also already covered in Section 3.3.2. The task may be converted to WFOMC using the conversion process described in Section 3.2.1, that is for every weighted formula $(\alpha_i, \lambda_i) \in \Phi$ a new equivalent formula is added, the new formula contains a new predicate ξ_i. By setting the weights of predicates to $w(\xi_i) = e^{\lambda_i}$, $\overline{w}(\xi_i) = 1$ and all remaining weights to 1, WFOMC of this augmented set of formulas is then equal to the partition function of MLN Φ. The numerator of the gradient for each α_i is then calculated in similar fashion as WFOMC($\Phi \cup \alpha_i\vartheta$, w, $\overline{w}$) for injective grounding substitutions ϑ of the formula α_i.

The relation between RMP and optimal solution to the weight learning problem is established in Lemma 16 in the article of Kuželka, Kungurtsev:

Lemma 16 from [14]: "*Let Φ be a set of quantifier-free first-order logic formulas, let Ω be a set of possible worlds and $A^{=}x = c$ be a maximal system of linearly independent equations satisfied by the vertices of the relational marginal polytope $\mathbf{P_R} = \mathrm{RMP}(\Phi, \Omega)$. Let θ be a point in the η-interior of $\mathbf{P_R}$. Then there is an optimal solution λ^* of the dual problem (3.7)*[1] *such that $A^{=}\lambda^* = 0$ and any such solution satisfies $\|\lambda^*\| \leq \log|\Omega|/\eta$.*"

The lemma provides two reasons why the knowledge of the relational marginal polytope is important for solving the weight-learning task:

1. the optimal solution satisfies equality $A^{=}\lambda^* = 0$ and the matrix $A^{=}$ may be obtained from equations that describe the RMP,
2. the upper bound for magnitude of the learned weights is inversely proportional to the interiority of the target marginal probabilities vector θ in the RMP.

The point 3 — optimization of the weights — is a constrained convex optimization problem. Ellipsoid algorithm is considered in the proofs provided in the article due to its favourable theoretical complexity properties, but for practical purposes authors recommend using projected gradient descent method.

Algorithm for weight learning in MLN then may be summarized in high-level as:

1. Calculate the RMP $\mathbf{P_R}$
2. Obtain matrix $A^{=}$ from $\mathbf{P_R}$
3. Using selected method for constrained convex optimization, maximize the dual criterion:

$$\max \sum_{\alpha_i} \lambda_i \cdot \theta_i - \log \sum_{\omega \in \Omega} e^{\sum_{\alpha_i} \lambda_i \cdot Q_\omega(\alpha_i)} \quad \text{s.t.}$$

$$A^{=} \cdot \lambda = 0$$

[1] Reference to the equation in this thesis plugged

Chapter 4
Implementation

This section describes programmatical implementation of the thesis and experiments performed on different versions of designed (I)RMP solvers — naive implementation of the domain lifted AISTATS algorithm, implementation of the algorithm with a tweaked WFOMC oracle and finally integer linear programming (ILP) based solvers. All modules are implemented in Python 3.6.9 if not stated otherwise.

4.1 Domain-lifted algorithm

The algorithm presented in Section 3.3.4 was implemented as a baseline for comparison with other implemented algorithms. The script is executable from file `aistats.py` in `polytopes` module. In order to implement the algorithm we need to resolve three problems:

1. calculation of a perpendicular vector to a m-tuple in higher dimensions,
2. calculation of the partition function,
3. enumeration (and an efficient enumeration) of normal vectors to be checked.

Calculation of normal vector from m-tuple of points consists of two steps. At first step one of the points is arbitrarily selected and subtracted from the others, so there are $(m-1)$ possibly linearly independent points remaining. Finding a perpendicular vector to these points in 2 and 3 dimensions (i.e. for MLNs containing 2, resp. 3 formulas) is performed by standard means — in 2D case the elements of the only remaining vector are simply swapped and one of them is negated, in 3D case the perpendicular vector is computed as a cross product of the two remaining points/vectors. In higher dimensions the following definition of generalized vector cross product based on calculation of subdeterminants is used, i.e. for $(n-1)$ (linearly independent) vectors in n dimensions we get:

$$\times(\vec{a_1},\ldots,\vec{a_{n-1}}) \equiv \det \begin{bmatrix} \vec{i_1} & \vec{i_2} & \ldots & \vec{i_n} \\ a_{1,1} & a_{1,2} & \ldots & a_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1,1} & a_{n-1,2} & \ldots & a_{n-1,n} \end{bmatrix}$$

Elements in the first row of the matrix are vectors of the standard basis. Somewhat interesting fact is that even Python *scipy* library lacks a method for generalized cross product[1], so in order to perform experiments a naive implementation was created which calculates the product exactly as in the definition.

Example 4.1. (4D cross-product calculation) *As a validation example let's calculate normal vector to plane defined by 3 points in 4 dimensions according to generalized cross product definition*

$$\vec{a_1} = (1,2,3,4), \quad \vec{a_2} = (0,-1,-2,-3), \quad \vec{a_3} = (-1,0,2,3)$$

[1] Probably because there is no universally accepted definition. Wikipedia article lists 8 different generalizations of cross product.

$$\times(\vec{a_1},\vec{a_2},\vec{a_3}) = \det\begin{bmatrix} \vec{i_1} & \vec{i_2} & \vec{i_3} & \vec{i_4} \\ 1 & 2 & 3 & 4 \\ 0 & -1 & -2 & -3 \\ -1 & 0 & 2 & 3 \end{bmatrix} = \vec{\mathrm{i}_1} \cdot \det\begin{bmatrix} 2 & 3 & 4 \\ -1 & -2 & -3 \\ 0 & 2 & 3 \end{bmatrix} -$$

$$- \vec{i_2} \cdot \det\begin{bmatrix} 1 & 3 & 4 \\ 0 & -2 & -3 \\ -1 & 2 & 3 \end{bmatrix} + \vec{\mathrm{i}_3} \cdot \det\begin{bmatrix} 1 & 2 & 4 \\ 0 & -1 & -3 \\ -1 & 0 & 3 \end{bmatrix} - \vec{\mathrm{i}_4} \cdot \det\begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & -2 \\ -1 & 0 & 2 \end{bmatrix} =$$

$$= (1,0,0,0) \cdot 1 - (0,1,0,0) \cdot 1 + (0,0,1,0) \cdot (-1) - (0,0,0,1) \cdot (-1) = (1,-1,-1,1)$$

Indeed we may easily check that the result is perependicular to each of the input vectors, so it is a normal vector to the hyperplane defined by these three points. We may also observe that all elements of the result are integers, which will hold in general as all input vectors considered in the algorithm are integers too. As noted in the description of the algorithm, in addition to an auxiliary MLN constructed from vector $(1,-1,-1,1)$ we also have to consider an MLN constructed from the opposite vector so the bounds of IRMP from the other half-space defined by the hyperplane are not omitted. The implementation also divides all elements of the vector by their greatest common divisor so they may be easily stored and retrieved from a lookup table. This way we may avoid unnecessary repeated calculations for integral multiples of the same vector.

Calculation of the partition function is converted to the WFOMC problem. *Forclift* [16][17] program for first-order knowledge compilation was employed as a WFOMC oracle inside the domain-lifted algorithm. The program internally compiles formulas into FO d-DNNF (*First Order determenistic Decomposable Normal Form*) circuit. Inference in the circuit takes polynomial time in relative to its size. This property enables Forclift to perform relatively fast inference, calculation of marginal probabilities for formulas and first of all calculation of the partition function partition functions especially for domain-lifted instances, where the size of the circuit remains polynomial w.r.t. the size of the domain. Forclift is implemented in Scala and its CLI provides option for querying the value of partition function for a first-order theory specified in a file. Unfortunately the program is designed to process only one file and terminate, so when using the vanilla version a new process must be created for each call to the oracle.

The essential key to the performance of the domain-lifted algorithm is the number of calls to the oracle as this is the most time consuming operation. The algorithm as stated checks every possible normal vector for points located inside the maximal bounding polytope. The baseline implementation that was used in experiments follows the same scheme, only keeping also a set of already checked vectors to avoid unnecessary repetitive evaluations on the same input.

4.1.1 Forclift wrapper

Executing a new Forclift process for each call to WFOMC oracle proved to be a performance limiting factor. Forclift is executed in Java Virtual Machine (JVM) and in order to fully utilize runtime optimization capabilities of JVM, a Java wrapper around Forclift was implemented. The utility is found in `forclift-wrapper` subdirectory of the source folder. It is implemented as a server over Netty [18] framework. The wrapper keeps JVM running and accepts name of files to be processed by Forclift, which are then passed directly into Forclift internal classes instead of its CLI.

In Python, the client side was implemented with standard `socket` library. The interprocess communication may be further improved by implementing asynchronous calls on client side, as server side is already prepared for handling multiple requests, allowing

parallelization of calculations, and finding another way of passing the models to Forclift than by standard files.

Forclift server may be started directly from the command line or implicitly by executing the main file of Python domain-lifted algorithm implementation `ai_main.py`. For starting the Forclift server directly, use command

```
java -jar forclift-rmp-wrapper-1.0.jar [PORT]
PORT - optional specification of port (default 8080)
```

Python file `ai_main.py` starts the server implictly on a random port as default if user doesn't specify otherwise. Such instance of the server is shut down when the program terminates. If user specifies a port number, Python assumes that the server is running on the specified port and tries to connect there. If the attempt fails, the process terminates (i.e. it doesn't even try to start the server again).

4.2 Integer linear programming solvers

This section describes another class of implemented RMP solvers based on integer linear programs checking realizibality of marginals of an MLN. As solving ILP is NP-hard task in general, these solvers cannot provide polynomial running time guarantees, but in practice – as will be shown in Experiments section – their performance proved to be superior over AISTATS algorithm.

4.2.1 Realizability of statistics

As the first step towards designing more efficient algorithms, an ILP for checking realizability of formula groundings number vector $\mathbf{N} = (n_1, n_2, ..., n_k)$ for MLN Φ and specified domain size $|\Delta|$ was formulated. The program decides if there exists a possible world ω with number of groundings $N(\alpha_i, \omega) = n_i$ and in positive case it also provides variable assignment for the world found. Input to the program is the domain size $|\Delta|$, a list of function-free quantifier-free first-order formulas Φ in CNF, and the vector of expected number of grounding for the formulas.

In the formulation we will use following symbols (sets and functions):

- $IVars$ — IRMP model variables
- $Pred$ — set of grounded predicates (i.e. every variable is substituted by an element of Δ)
- Cl — set of unique clauses among all formulas, clauses are considered equivalent if it is possible to establish a one-to-one injective mapping between variable names
- $Cl_{k,\vartheta}$ — set of all possible variable substitutions ϑ of clause $c_k \in Cl$
- $CP : Cl_{k,\vartheta} \to IVars$ — map from grounded clause to model variables associated with literals of the grounded clause
- $\Theta(\alpha_i)$ — set of all variable substitutions $\alpha_i\vartheta$ of formula α_i
- $FC : \Theta(\alpha_i) \to IVars$ — map from grounded formula to model variables associated with clauses of the grounded formula

ILP 4.1. (*Realizability checking ILP*) Using these sets we may define the ILP:

$$\max 0 \quad \text{s.t.} \tag{4.1}$$

$$\forall p \in Pred: \; v_p + \overline{v_p} = 1, \; v_p \in \{0,1\}, \; \overline{v_p} \in \{0,1\} \tag{4.2}$$

$$\forall c_{k,\vartheta} \in Cl_{k,\vartheta}: \; C_{k,\vartheta} = \max\{CP(c_{k,\vartheta})\} \tag{4.3}$$

$$\forall\, \alpha_i\vartheta \in \Theta(\alpha_i):\ A_{i,\vartheta} = \min\{FC(\alpha_i\vartheta)\} \tag{4.4}$$

$$\forall\, n_i \in \mathbf{N}:\ \sum_{\vartheta} A_{i,\vartheta} = n_i \tag{4.5}$$

Such definition looks a little bit complicated, but the following description of each equation hopefully makes it clearer:

1. The optimization criterion (4.1) is just a constant function as the ILP only checks feasibility of constraints.
2. On line (4.2) two binary variables are created for every possible ground atom in Φ, variable v_p indicates if the grounding is true in current interpretation and variable $\overline{v_p}$ is its negation. The condition $v_p + \overline{v_p} = 1$ simply ensures that only one of these variables is true at time. The negated variable is just added for implementation convenience and we could use term $(1 - v_p)$ instead as well.
3. Equation (4.3) calculates boolean value of clause under variable substitution ϑ. Clause is disjunction of literals therefore the boolean value is equal to maximum of all variables representing the literals. As an example let's consider clause $smokes(X) \vee \neg\, friends(X, Y)$ and a substitution $\vartheta : \{X \to John, Y \to Paul\}$. The constraint then achieves a form:

$$C_{i,\vartheta} = \max\{v_{smokes(John)}, \overline{v_{friends(John,Paul)}}\}$$

4. Analogically Equation (4.4) calculates boolean value of the whole formula α_i under variable substituion ϑ. The formula is represented as conjunction of clauses so its boolean value is equal to the minimum among all the variables representing values of the clauses (under substitution ϑ). Using the clause and substitution from previous point and by adding another clause $smokes(Y) \vee \neg\, friends(X, Y)$ we get:

$$\alpha_i:\ (smokes(X) \vee \neg friends(X, Y)) \wedge (smokes(Y) \vee \neg friends(X, Y))$$

$$\begin{aligned} C_{j,\vartheta} &= \max\{v_{smokes(John)}, \overline{v_{friends(John,Paul)}}\} \\ C_{k,\vartheta} &= \max\{v_{smokes(Paul)}, \overline{v_{friends(John,Paul)}}\} \\ A_{i,\vartheta} &= \min\{C_{j,\vartheta}, C_{k,\vartheta}\} \end{aligned}$$

5. Finally in (4.5) the total number of satisfied groundings per each formula is set equal to corresponding value of the input vector.

An example of full definition of IRMP for statistics feasibility checking over two formulas on the domain of size two is provided in Appendix C.

Gurobi [19]) is used as ILP solver in the implementation. When the solver checks feasibility of generated model and finds no violation of constraints, it also returns an assignment of all variables which in turn represent one of possible worlds with specified formula groundings vector $\mathbf{N}$ — we should note that the assignment isn't necessarily unique and there may be substantially more possible worlds satisfying the same set of constraints. We should also stress that this program doesn't perform *containment test*, i.e. it doesn't decide whether the point corresponding to specified number of groundings is inside the IRMP for the MLN Φ. The IRMP (Equation (3.9)) is defined as a convex hull of all feasible formula grounding counts, therefore even if the ILP model is evaluated as infeasible by the solver, the point may be still contained in the IRMP. But the negative result still provides us at least some information — in such case we can conclude the point is definitely not a vertex of the IRMP.

The model generally creates $\mathcal{O}(|\Delta|^k)$ ground atom variables (where k represents maximum number of variables in any predicate, i.e. $k = \max\{|vars(Pred_i)|\}$) for every possible ground substitution and similar asymptotic bounds based on maximum number of varibales in a clause/formulas hold for number of model variables associated with clauses and formulas. The number and size of constraints is also polynomially bounded by the domain size and maximum number of variables present in respective conditions. However, even though the size of the model is polynomial in $|\Delta|$, it grows rather steadily. In addition solving ILP is an NP-hard problem, so from theoretical point of view algorithms based on ILP cannot provide more favourable guarantees even for domain-liftable MLNs to the algorithm based on WFOMC reduction.

Python implementation of this ILP is stored in file `possible_world.py` in `polytopes` module. It actually provides slightly more features, which are documented in the code or described in the following sections.

4.2.2 Improvements to ILP definition

The definition of ILP presented in previous section provides a tool for checking realizability of arbitrary formula groundings count vector, but it is not really helpful for constructing the IRMP. We may actually construct the IRMP by solving the ILP for each of $\mathcal{O}(|\Delta|^{\prod |vars(\alpha_i)|})$ points in the maximal bounding polytope, which will output a list of all possible worlds, but the number of calls will obviously be enormous. In this subsection modifications of the ILP definition leading to more efficient calculation of IRMP will be discussed.

The first modification is related to Equation (4.5). We may easily see that we don't have to restrict the constraints just to equalities, but we may also include inequalities. By this generalization we may answer more questions related to the formula grounding vector just by changing the last set of constraints, for example:

- given the vector $\mathbf{N} = (n_1, ...n_k)$ and keeping all but one element n_i fixed, is there a possible world for such formula grounding count?

$$\forall\, n_j \in \mathbf{N},\ n_i \neq n_j : \sum_{\vartheta} A_{j,\vartheta} = n_j$$

$$n_i = \sum_{\vartheta} A_{j,\vartheta}$$

  (here n_j are integers and n_i is variable, if the model is feasible we may retrieve its value from the solver in the same way as with boolean values for predicate assignment substitution),
- or given the vector $\mathbf{N}$, we may split its elements into two disjunct sets $N_=$ and $N_\leq$ and check feasibility of the model with equality constraints for elements in $N_=$ and inequality constraints for elements in $N_\leq$:

$$\forall n_i \in N_= : \sum_{\vartheta} A_{i,\vartheta} = n_i$$

$$\forall n_k \in N_\leq : \sum_{\vartheta} A_{k,\vartheta} \leq n_k$$

- and any possible combination of equalities and non-strict inequalities for each element of $\mathbf{N}$ independently, or even introducing interval checks on n_i (i.e. is there a possible world with number of groundings for formula α_i in interval $[l_i,u_i]$?):

$$\forall n_i \in \mathbf{N} :\ l_i \leq \sum_{\vartheta} A_{i,\vartheta} \leq u_i$$

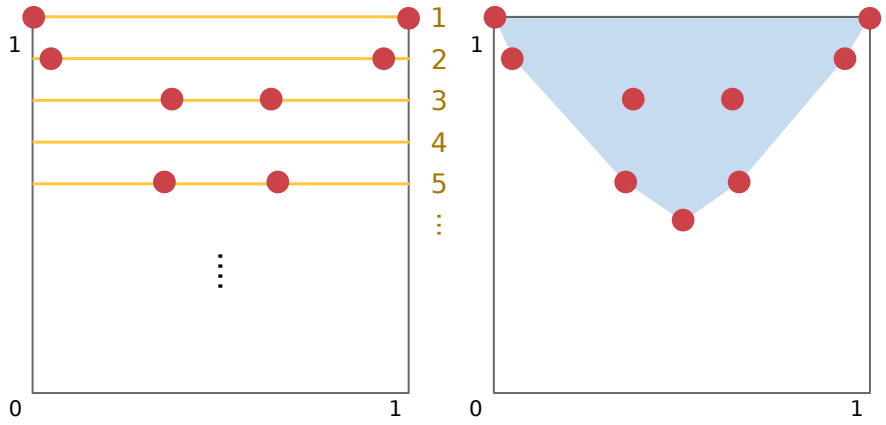

Figure 4.1. **Left** Illustration of execution of the "cutting" ILP algorithm based on fixing all but one coordinates for the IRMP in 2D case (i.e. for two formulas). Orange lines represent each cut and we may imagine as proceeding in order from line 1, red points represent the extramal coordinates for possible world in each cut. Note that the extremal points found in a cut may not necessarily be vertices nor even lie on the boundary of the polytope (line 3) and there may be even gap (line 4), i.e. a cut where no possible world is found. **Right** After running the algorithm for all cuts, we may construct the IRMP as a convex hull of all discovered extremal points. In this case, only one more vertex was found.

ILP 4.2. (*"Cutting" ILP*) The first point in the previous listing actually leads to another possible modification of the ILP. If we consider all but one element of **N** to be fixed (again denoting the free element as n_i), we may also search not only an arbitrary possible world with admissible value for n_i but also possible worlds with maximal or minimal value for n_i just by adding the variable to the optimization criterion function and set the goal to maximize/minimize it:

$$\max/\min \quad n_i$$

This modification of the ILP leads to another — slightly less naive — algorithm for calculation of the IRMP. Instead of execution for every possible point in the maximal bounding polytope we may just iterate over all posible values for vector $(x_1, ..., x_{n-1})$ from fixed subset of $n-1$ coordinates and for every such cut find the maximal and minimal value for the n-th coordinate by executing the ILP twice for each optimization sense separately. Illustration of this procedure for 2D case is shown in Figure 4.1. To minimize the number of calls we should leave out the dimension with the highest maximal possible number of groundings. Nevertheless this algorithm doesn't provide any substantial improvement in execution speed. There also exists a possibility that no feasible solution is found for the current cut even when the cut actually intersects the IRMP, so in order to exactly compute the whole IRMP we cannot perform some sort of fast-circuit logic and cancel the execution when such a cut is visited (this is also illustrated in Figure 4.1). On the other hand, if we don't need to calculate the IRMP exactly, we may easily change this algorithm to heuristic one just by skipping some cuts during the execution either randomly or by application of some rule (e.g. skip odd values or check only values divisible by some integer).

4.3 Convex hull algorithm

This section proposes an algorithm based on ILP calculation constructing the IRMP by iteratively searching a new point belonging to the polytope which achieves maximal

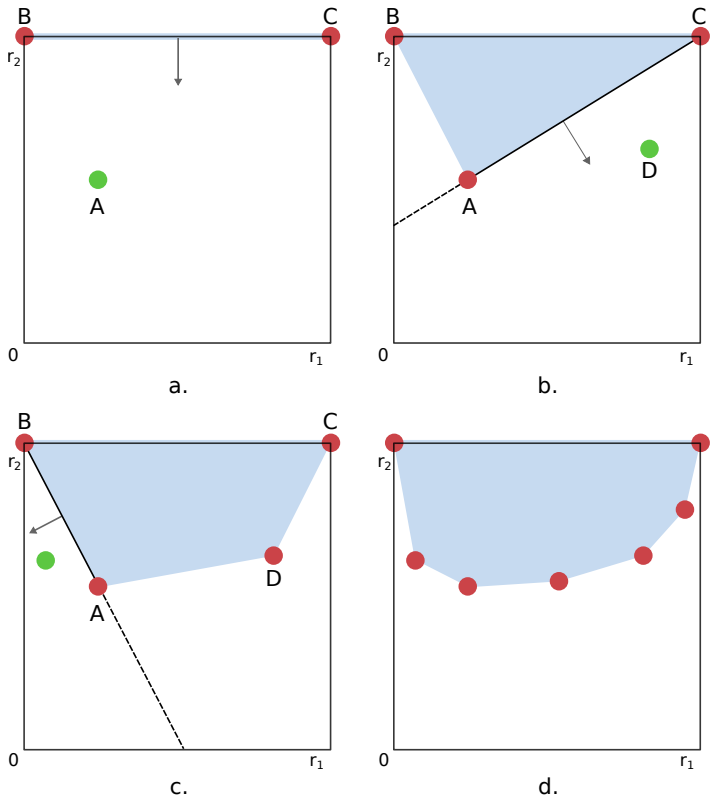

Figure 4.2. Illustration of execution of the ILP–convex hull algorithm on 2D case (i.e. for an MLN with two formulas). **a.** Initial vertex search identified corner vertices B, C as members of the IRMP. Edge BC (facet in higher dimensions) is pushed to the queue, immediately popped and the point with the maximal distance from the edge in the direction indicated by the grey arrow is calculated by ILP. The new point A is added to the convex hull and the two new edges are pushed to the queue. **b.** Edge AC is popped from the queue and the ILP is executed again to find next IRMP vertex D. **c.** The algorithm continues with processing edge AB. **d.** Finally (with many steps skipped) when the queue is empty, we may conclude that the whole IRMP was found.

distance from a facet of the current polytope. In this sense the algorithm proceeds in opposite direction to AISTATS algorithm described in Section 3.3.4 — while AISTATS algorithm starts from the maximal bounding polytope and iteratively restricts its size until no more conditions may be created (i.e. until all normal vectors are enumerated and checked in the naive implementation), the proposed algorithm builds the IRMP bottom-up. The polytope state in each step of the algorithm represents current lower bound on the actual IRMP range and the algorithm gradually tightens the gap until no more vertices could be found.

The algorithm internally stores current set of discovered vertices of the polytope and for searching a new vertex it uses hyperplane representation of polytope's facets, so we need to switch between both V- and H-representation of the polytope. *Qhull* [20] library, resp. its Python port in `scipy` library, is used for manipulating the polytopes.

The algorithm also requires calculation of distance between a point and a facet of the current polytope. H-representation of the polytope comes handy in this case as we can then easily retrieve the equation of the hyperplane defining the facet. The equation in Qhull is stored as $\vec{a} \cdot \vec{x} + b \leq 0$. Using standard formula for point-to-hyperplane distance we can calculate the distance between point $\vec{y}$ and its nearest point in the facet $\vec{x}$ as:

$$\|\vec{x} - \vec{y}\|_2 = \frac{|\vec{y} \cdot \vec{a} + d|}{\|\vec{a}\|_2}$$

In the defintion of ILP which follows immediately we may actually omit the denominator as we consider only one facet at time, so the norm of vector $\vec{a}$ remains constant and selection of the furthest point depends only on the value of the numerator.

ILP 4.3. (*ILP finding possible world furthest from a hyperplane*) The task of finding a point representing a possible world with maximal distance from a facet of polytope represented by the hyperplane $\vec{a} \cdot \vec{x} + b \leq 0$ for MLN Φ is solved by ILP (using notation from the listing on the page 26):

$$\max d \quad \text{s.t.} \tag{4.6}$$

$$\forall\, p \in Pred: \;\; v_p + \overline{v_p} = 1, \;\; v_p \in \{0,1\}, \;\; \overline{v_p} \in \{0,1\} \tag{4.7}$$

$$\forall\, c_{k,\vartheta} \in Cl_{k,\vartheta}: \;\; C_{k,\vartheta} = \max\{CP(c_{k,\vartheta})\} \tag{4.8}$$

$$\forall\, \alpha_i\vartheta \in \Theta(\alpha_i): \;\; A_{i,\vartheta} = \min\{FC(\alpha_i\vartheta)\} \tag{4.9}$$

$$\forall\, \alpha_i \in \Phi: \;\; x_i = \sum_{\vartheta} A_{i,\vartheta} \tag{4.10}$$

$$b + \sum_i a_i \cdot x_i \;\geq 0 \tag{4.11}$$

$$d = \;|\, b + \sum_i a_i \cdot x_i \,| \tag{4.12}$$

In the criterion function (Equation (4.6)) we maximize variable d which is proportional to distance of a point to the facet of the current polytope, calculation of this variable is captured in the last Equation (4.12). Equations (4.7) through (4.9) represent the theory Φ and are actually same as equations (4.2)–(4.4) from the ILP for feasibility checking. Equation (4.10) stores current number of true groundings for formula α_i into variable x_i and Equation (4.11) ensures the search is performed in the half-space complementary to the one defined by the hyper-plane inequality. The assignment to varibles x_i for the optimal value of d then corresponds to coordinates of the possible world which is furthest from the facet.

The penultimate equation (4.11) should formally be a strict inequality so the ILP doesn't consider points located exactly on the facet, but *gurobi* solver supports non-strict inequalities only. If the coefficients describing the hyperplane are integers, the constraint can be substituted by inequality

$$b + \sum_i a_i \cdot x_i \;\geq 1$$

as then both b and the scalar product $\langle a, x \rangle$ are integers. However *Qhull* library internally normalizes the normal vectors of H-representation to unit length, so the coefficients are floating point numbers and such fix is unapplicable in this case. On the other hand, due to the normalization the value of d in the implementation is actually real distance between the point and the facet, even though the denominator is omitted in the ILP formulation. A constraint enforcing that d is greater than some threshold then may be added to the model to exclude points on the facet. The threshold may be chosen either arbitrarily or calculated. The calculation requires reconstructed vector $\vec{a'}$ of integer coefficients of the hyperplane's normal vector from the normalized vector $\vec{a}$ (the greatest common divisor of the reconstructed coefficients must be equal to 1). The

maximal value of the threshold is then equal to the minimal *non-zero* distance between a hyperplane with normal vector $\vec{a'}$ and an integer point. This distance is proportional to inverse of vector's norm [21] and limits on the threshold ε are:

$$0 < \varepsilon < \frac{1}{\|\vec{a'}\|_2} (\leq d)$$

The last step that must be resolved before the actual description of the algorithm is creation of initial polytope, because *Qhull* requires as an input definition of at least $(n + 1)$ non-colinear points for specification of a polytope in the space of dimension n. The points may be generated in multiple ways:

1. the previously specified ILP 4.3 may still be employed using facets of the maximal bounding polytope instead (i.e. hyperplanes with equations $x_i = 0$ or $x_i = r_i$, where r_i is number of possible injective groundings of formula α_i), it may however happen that the furthest point for multiple facets will be the same,
2. the ILP 4.2 for detecting the maximum/minimum coordinates for specified cut may be used until $(n + 1)$ required points are found,
3. or we may check satisfiability in vertices ("corners") of the maximal bounding polytope

The implementation uses process described in the last point. For this task we could employ the very first ILP 4.1 for checking realizability of formula statistics at the selected point, but we may also exploit the fact that the vertices of the maximal bounding polytope actually represent extremal statistics for achievable number of groundings. Instead of calling ILP over grounding of original the first-order theory, we may check satisfiability of derivated propositional theory which uses only predicate names, e.g. instead of predicate formula $\alpha : edge(X, Y) \vee \neg edge(Y, X) \vee foo(X)$ we create propositional formula $edge \vee \neg edge \vee foo$ (formula with multiple occurences of the same predicate name with different order of variables is selected intentionally to show that the variables are irrelevant in this particular problem). This trivial transformation can be performed because MLN formulas are both function- and quantifier-free. Assigning boolean value to the proposition *edge* may be interpreted as assigning the same boolean value to all possible groundings of predicate $edge(X, Y)$. When the propositional formula is satisfiable, so are all groundings in its predicate counterpart and therefore $N(\alpha_i, \omega) = r_i$, conversely when it is unsatisfiable then none grounding may be true and $N(\alpha_i, \omega) = 0$.

A SAT-solver can be employed for checking whether a corner vertex of the maximal bounding polytope is also a vertex of the IRMP, but in the implementation an ILP formulation is used again. The ILP model size in this case depends only on the number of formulas and distinct predicate names, so it is independent of the domain size.

ILP 4.4. (*Corner checking ILP*) The following ILP checks if vertex $V = (v_1, ..., v_n)$, $v_i \in \{0, r_i\}$ (here r_i again denotes the maximum number of possible groundings of formula α_i) of the maximal bounding polytope represents a possible world by checking propositional theory Φ' created from predicate theory Φ:

$$\max 0 \quad \text{s.t.}$$

$$\begin{aligned} &\text{for each atom } p: \; v_p + \overline{v_p} = 1, \; v_p \in \{0,1\}, \; \overline{v_p} \in \{0,1\} \\ &\quad \text{for each clause } c: \; x_c^+ = \max\{v_p \text{ if } p \text{ in } c\} && (4.13) \\ &\quad \text{for each clause } c: \; x_c^- = \max\{\overline{v_p} \text{ if } \neg p \text{ in } c\} && (4.14) \end{aligned}$$

$$\text{for each clause } c: \; x_c = \max\{x_c^+, x_c^-\} \tag{4.15}$$
$$\text{for each formula } \alpha_i: \; A_i = \min\{x_c \text{ if } c \text{ in } \alpha_i\} \tag{4.16}$$
$$\text{for each } v_i = 0: \; A_i = 0 \tag{4.17}$$
$$\text{for each } v_i > 0: \; A_i \geq 1 \tag{4.18}$$

The definition is indeed very similar to the realizability checking ILP 4.1, differences arise due to the fact that a propositional theory is used in this case so we don't need to evaluate assignments to all possible variable substitutions. Equations (4.13) to (4.15) describe truth value of each clause, equation (4.16) is again a boolean value of the whole formula α_i. Last two equations connect the truth value of formula with expected number of groundings — it should be zero for vertex coordinates equal to zero and non-zero for the others (i.e. coordinates equal to r_i). If the constraints of the ILP are feasible, the vertex in question is actually also a vertex of the IRMP.

In order to find the initial polytope for n formulas we just need to iterate over corners of the maximal bounding polytope until $(n+1)$ points satisfying the ILP are found. If all corners 2^n are exhausted and the number of vertices for the initial polytope still didn't reach $(n+1)$, we have to resort to another methods — our implementation then uses algorithm based upon "cutting" ILP 4.2 until the required number of initial points is detected.

Finally we can describe the algorithm for calculation of IRMP for MLN Φ using following two methods:

- **GET_INITIAL_HULL** — method for creating the initial polytope (by combination of ILP 4.4 and ILP 4.2 if necessary)
- **FURTHEST_FROM_FACET** — method for finding coordinates of possible world not included in the current polytope with the greatest distance from the selected facet of the current polytope (by ILP 4.3)

The algorithm then proceeds in following steps:

Algorithm for calculation of IRMP

```
input: (MLN Φ, domain size |Δ|)
1   queue = ∅
2   IRMP ← GET_INITIAL_HULL(Φ)
3   queue = {facets of IRMP}
4   until queue is empty ; do
5     F ← queue.pop()
6     P ← FURTHEST_FROM_FACET(F)
7     IRMP ← IRMP ∪ {P}
8     F' ← {new facets of IRMP}
9     queue.push(F')
10  done
11  return IRMP
```

If it is not necessary to calculate the IRMP exactly, the algorithm may be easily changed to heuristic one. One possible way how to achieve this is to introduce some randomization parameter ρ which will decide whether to execute calculation for current facet retrieved from the queue or to skip it. Although we are not able to provide rigorous guarantees about such approximation, Figure 4.3 illustrates that given the approximation of the IRMP and a list of facets that were skipped and not further

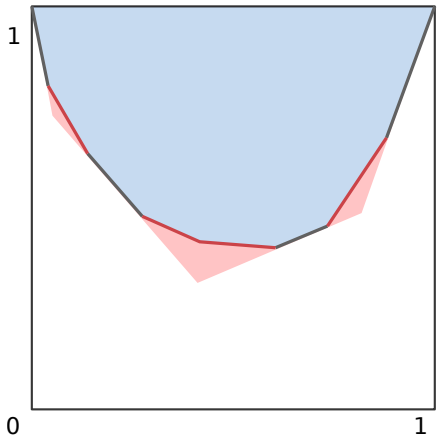

Figure 4.3. Illustration of the extent of maximal error for the heuristic version of the main algorithm that. The IRMP (or more precisely its approximation) is highlighted in blue, edges which were skipped are marked red and the area of possibly unexplored vertices of the IRMP may is highlighted in pink.

explored during the execution we may actually bound the maximal extent of the area where an unexplored vertices could be present. The area is bounded by hyperplanes of the *unskipped* facets, so it is another convex polytope. While the result returned by the randomized algorithm represents the lower bound of the IRMP, the polytope obtained from the unskipped facets is actually the upper bound. In other words — execution of the algorithm that is intended to provide lower bound for the IRMP simultaneously yields also the upper bound (if we store list of (un)skipped facets). It is also somewhat interesting that while the lower bounding polytope is stored in V-representation as a convex hull of potential vertices, the upper bounding polytope is retrieved from H-representation of all *closed* facets (i.e. facets that were already processed and no possible world in the complementary half-space was found), indicating some sort of duality between these representations.

4.4 Experiments

This section evaluates performance of the implemented algorithms for calculation of the RMPs over some MLNs. Experiments were executed on a laptop with 4GB RAM, 4 × Intel i3 2,2 GHz CPU running on Ubuntu 18. Some figures and tables generated from experiments are located in Appendix D in order to not filling the following pages by an unnecessary amount of images.

As a first case we will use two-formula MLN *knows and likes*:

- 1 $likes(X, Y) \vee \neg knows(X, Y)$
- 3 $\neg knows(X, Y) \vee \neg likes(X, Y) \vee friends(X, Y)$

This optimistic example could be interpreted as a representation of the world where 1. everyone likes all people they know and if they do not like someone, it's just because they do not know them, 2. if you know and like someone, the you are a friend with them.

Images in Figure 4.4 show that ILP is indeed faster than the naive algorithm by an order of one or two magnitudes (the time scale in the Top image is logarithmic). We may also see that execution of Forclift oracle in server mode also substantially improves runtime. Bottom image shows the output of the naive algorithm and we may see it is not

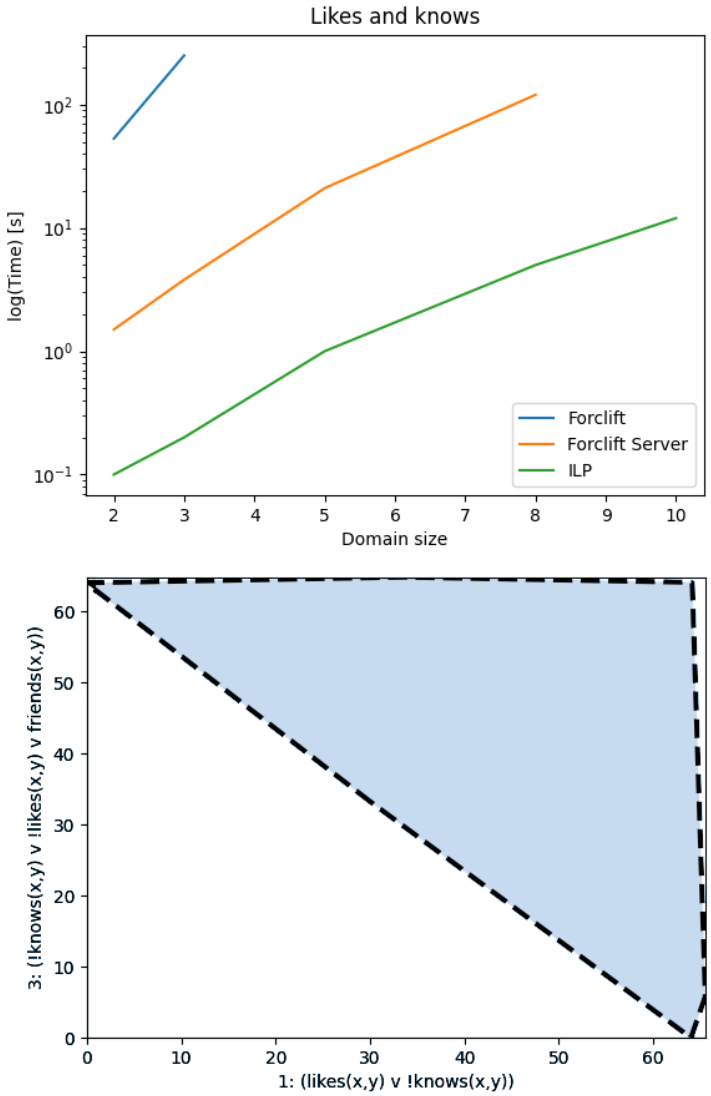

Figure 4.4. **Top** Running times of *likes and knows* MLN according to the domain size. Note the time scale is logarithmic. Data in tabular form may be found in Table D.1. **Bottom** Inaccurate representation of IRMP for $|\Delta| = 8$ calculated by the implementation of the naive algorithm. The true IRMP should be just a triangle with vertices [0,64], [64,64] and [64,0] (in this particular execution the constraint $X \neq Y$ was omitted).

right, as the actual RMP should be actually a triangle connecting all but one corners. This is probably due to the fact that partition function of the auxiliary polytopes is too high and calculation of coefficient b becomes too imprecise. The ILP implementation returns the polytope correctly in significantly shorter time (see Figure D.1).

As a second test example following *friends* MLN was considered:

- 1 $friends(X, Y)$
- 1.2 $\neg friends(X, Y) \vee friends(X, Z) \vee friends(Y, Z)$

As the second formula contains three free variables, it is not domain liftable so only the ILP algorithm was tested.

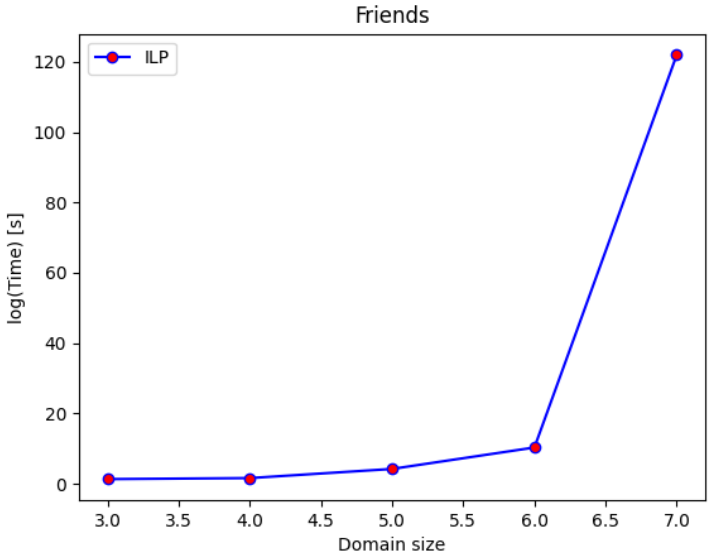

Figure 4.5. Execution time for *friends* MLN for domain sizes 3–7.

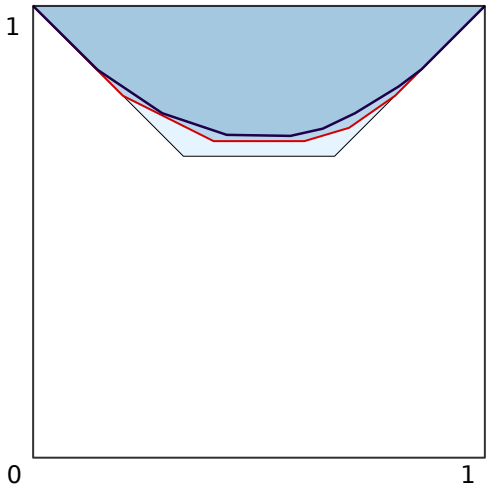

Figure 4.6. RMP for *friends* MLN and domain size 7 (purple), 5 (red) and 3 (black). We can see that the convex hull is gradually becoming more complex and also that RMP for larger domains are indeed subsets of those for smaller one. RMPs for sizes 4 and 6 are in Appendix D.

In Figure 4.5 we see that execution time rises sharply for the domain size of 7. Domain sizes greater than 7 couldn't be assessed as the number of variables in the ILP exceeds the maximum of the *gurobi* license.

Figure 4.6 confirms that RMP over the same set of formulas for larger domains are subsets of RMPs for the smaller domains, i.e. in the figure the largest polytope corresponds to domain size 3 and volume of RMP then gradually shrinks.

Chapter 5
Conclusion

The main goal of the thesis was designing efficient algorithms for construction of relational marginal polytopes (or their integer counterparts, which are — as was established in the work — equivalent). The introductory part of the work described basic terminology and principles related to the first-order (predicate) logic and also discussed possible extensions of ordinary bivalent logic for capturing uncertainty or probabilistic information. As an addendum to this, two classes of commonly used probabilistic graphical models were also briefly described — Bayesian networks and Markov networks. The latter model is directly related to the main topic of the thesis, Markov logic networks (MLN). On the other hand the former may be considered highly unrelated to MLNs, but its description was incorporated into the thesis due to the fact that it receives significantly more space in the curriculum of the study programme, so it was rather natural to describe Markov networks by comparison to the Bayesian ones.

Chapter 3 formally introduced Markov logic network, which is a probabilistic logic based on ideas of Markov networks applied into first-order logic. As was remarked, the MLN may be considered a template for creating Markov networks which are different in size but share similar structures. The chapter followed with definition of the relational marginal problem, the task of finding the maximum entropy probability distribution over the possible world of the MLN satisfying requirements on marginal probabilities for formulas in the MLN. The main topic of the thesis — relational marginal polytopes (RMP)— stems from the relational marginal problem as it represents the set of feasible marginal probabilities. An algorithm for constructing the RMP for domain-liftable MLNs is described and finally a relation between solution of the maximal-likelihood weight learning task and the RMP is described.

In chapter 4 (Implementation) a number of algorithms for construction of relational marginal polytopes was described and two of them — a baseline domain-lifted algorithm based on calls to WFOMC oracle and an algorithm based on ILP for finding coordinites of a possible world with maximal distance relative to a hyperplane — were compared. As was anticipated, the baseline algorithm based was overperformed by ILP based exact solver. In respect to the goal of designing efficient heuristics, two possible heuristic criterions for deciding whether to process or skip a facet were described. However, no rigorous guarantees were provided for the heuristics so neither of them were proved to be a proper approximation algorithm, but a method how to assess the limits of possible error was described. This is due to the fact that during the IRMP vertex search the V-representation of potential vertices happens to be a lower boundary of the IRMP and the H-representation of all processed and closed facets (closed in a sense that it was proved no possible world exists in the complementary hyperspace) is actually the upper boundary of the IRMP.

The last goal of the thesis — incorporating the IRMP construction into algorithm for maximum-likelihood weight learning of MLN or finding a way how to detect when the polytope is not fully-dimensional — was not fullfiled as of the date of submission. The main goal of the thesis was however satisfied.

In the ultimate paragraph we propose a few possible improvements to the work or future tasks. First, it would definitely be beneficiary to test the designed algorithms on a set of more challenging MLNs instances than those used in the experiments section. Author expects that if the experiments were performed on a more modern and powerful machine, the difference between the naive algorithm and the ILP algorithm would be even bigger, as the ILP solver should be probably able to benefit more from improvements in available resources. With respect to the naive domain-liftable algorithm, a more reasonable method for selection of normal vectors to be processed could be implemented, however it is rather questionable if it is worth the effort as the results of experiments show that the algorithm based on Forclift as a WFOMC oracle becomes inaccurate rather quickly even for relatively small instances. Another oracle — for example based on *Sententical Decision Diagrams* (SDD) [22] calculating the WFOMC as WMC of its the formula groundings — could be used instead, but it doesn't utilize domain-liftability properties of WFOMCs. And as the last remark it seems that the ILP algorithm could be also parallelized rather easily, as there are probably only three probably trivial race-conditions — pushing/popping facet from the queue and maybe update to the Qhull object which captures current state of the polytope.

References

[1] KUŽELKA, Ondřej and Yuyi WANG, 2020. Domain-Liftability of Relational Marginal Polytopes. In: *AISTATS 2020: 23rd International Conference on Artificial Intelligence and Statistics*. Accessible from: https://arxiv.org/pdf/2001.05198.pdf

[2] NILSSON, Nils J., 1986. Probabilistic logic. *Artificial Intelligence*. **28**(1), 71-87. DOI: 10.1016/0004-3702(86)90031-7. ISSN 0004-3702.

[3] ZADEH, L.A. Fuzzy sets. *Information and Control*. 1965, **8**(3), 338-353. DOI: 10.1016/S0019-9958(65)90241-X. ISSN 00199958. Accessible from: http://linkinghub.elsevier.com/retrieve/pii/S001999586590241X

[4] JAYNES, E. T., 1957. Information Theory and Statistical Mechanics. *Physical Review*. **106**(4), 620-630. DOI: 10.1103/PhysRev.106.620. ISSN 0031-899X. Accessible from: https://link.aps.org/doi/10.1103/PhysRev.106.620

[5] SHANNON, C. E., 1948. A Mathematical Theory of Communication. *Bell System Technical Journal*. **27**(4), 623-656. DOI: 10.1002/j.1538-7305.1948.tb00917.x. ISSN 0005-8580. Accessible from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6773067

[6] DAWID, A. Philip, 1980. Conditional Independence for Statistical Operations. *The Annals of Statistics*. **8**(3), 598-617. DOI: 10.1214/aos/1176345011. ISSN 0090-5364. Accessible from: http://projecteuclid.org/euclid.aos/1176345011

[7] PEARL, Judea and Stuart RUSSELL, 2002. Bayesian networks. *The Handbook of Brain Theory and Neural Networks* 2nd Edition. Cambridge (MA): MIT Press. ISBN 9780262011976.

[8] KOLESHOV, Volodymyr and Stefano ERMON, 2020. Bayesian networks. *Stanford CS 228 - Notes* [online]. [cit. 2020-08-14]. Accessible from: https://ermongroup.github.io/cs228-notes/representation/directed/

[9] PEARL, Judea, 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* . 1. Amsterdam: Elsevier, 552 pp. DOI: https://doi.org/10.1016/C2009-0-27609-4. ISBN 978-0-08-051489-5.

[10] Markov random field, 2020. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation [cit. 2020-08-14]. Accessible from: https://en.wikipedia.org/wiki/Markov_random_field#Definition

[11] KARP, Richard M., 1972. Reducibility among Combinatorial Problems. In: *Complexity of Computer Computations*. Boston, MA: Springer US, 1972, p. 85-103. DOI: 10.1007/978-1-4684-2001-2_9. ISBN 978-1-4684-2003-6. Accessible from: https://people.eecs.berkeley.edu/~luca/cs172/karp.pdf

[12] RICHARDSON, Matthew and Pedro DOMINGOS, 2006. Markov logic networks. *Machine Learning*. **62**(1-2), 107-136. DOI: 10.1007/s10994-006-5833-1.

ISSN 0885-6125. Accessible from:
http://link.springer.com/10.1007/s10994-006-5833-1

[13] VAN DEN BROECK, Guy, 2011. On the completeness of first-order knowledge compilation for lifted probabilistic inference. *Advances in Neural Information Processing Systems*. **24**, 1386—1394. Accessible from:
https://dl.acm.org/doi/10.5555/2986459.2986614

[14] KUŽELKA, Ondřej and Vyacheslav KUNGURTSEV, 2019. Lifted Weight Learning of Markov Logic Networks Revisited, In: *Proceedings of Machine Learning Research*, 89, p. 1753–1761. Accessible from:
http://proceedings.mlr.press/v89/kuzelka19a/kuzelka19a.pdf

[15] BEAME, Paul, Guy VAN DEN BROECK, Eric GRIBKOFF and Dan SUCIU, 2015. Symmetric Weighted First-Order Model Counting. *Proceedings of the 34th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. New York, NY, USA: ACM, 2015-05-20, 313-328. ISBN 9781450327572. DOI: 10.1145/2745754.2745760. Accessible from:
https://arxiv.org/abs/1412.1505

[16] WFOMC / ForcLift, 2016. Leuven: DTAI - KU Leuven. Accessible from:
https://dtai.cs.kuleuven.be/software/wfomc

[17] VAN DEN BROECK, Guy, Nima TAGHIPOUR, Wannes MEERT, Jesse DAVIS and Luc DE RAEDT, 2011. Lifted probabilistic inference by first-order knowledge compilation. In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*. 21, p. 2178–2185. DOI:10.5591/978-1-57735-516-8/IJCAI11-363. ISSN 1045-0823. Accesible from:
https://lirias.kuleuven.be/retrieve/150085

[18] Netty , 2020. Accessible from:
https://netty.io/

[19] Gurobi Optimization, LLC, 2021. *gurobi*. Gurobi Optimizer Reference Manual. Accessible from:
http://www.gurobi.com

[20] Qhull , 2021. Accessible from:
http://www.qhull.org/

[21] NICOLAS, André, 2012. Closest point to line in lattice. *Mathematics Stack Exchange* [online]. [cit. 2021-5-21]. Accessible from:
https://math.stackexchange.com/questions/100202/closest-point-to-line-in-lattice

[22] DARWICHE, Adnan, 2011. SDD: A New Canonical Representation of Propositional Knowledge Bases. In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*. Barcelona: AAAI Press, p. 819-826. ISBN 9781577355144. Accessible from:
https://dl.acm.org/doi/10.5555/2283516.2283536

Appendix A
List of abbreviations

CLI	command line interface
CNF	conjuctive normal form
FOL	first-order logic (also predicate logic)
ILP	integer linear programming
IRMP	integerrelational marginal polytope
JVM	Java Virtual Machine
KB	knowledge base
MLN	Markov logic network
MRF	Markov random field (also Markov network)
RMP	relational marginal polytope

Appendix B
Supplementary data and documentation

B.1 Source code

Source code of the thesis is publicly available at `https://github.com/kozakja4/m_thesis`

B.2 Content of the repository

```
root
 |_ Code ..... source code folder
 |   |_ python ..... source codes of python modules
 |   |_ java .....source code of java module forclift-wrapper
 |_ img .....figures including their TikZ or Python definitions
 |_ text.pdf ..... text of the thesis
```

Appendix C
Feasibility IRMP example

Check feasibility for statistics vector $\mathbf{N} = (1, 2)$ over domain $\Delta = \{Peter, John\}$ for formulas Φ:

- $\alpha_1 : \ edge(X, Y) \wedge (edge(Y, X) \vee friends(Y, X))$
- $\alpha_2 : \ \neg friends(X, Y) \vee \neg smokes(X) \vee smokes(Y)$

We're checking feasibility, so we formally optimize w.r.t. constant function.

$$\max 0 \quad \text{s.t.}$$

Create two binary variables (i.e. their value is either 0 or 1) for every possible grounding of predicates in Φ and set their sum to be equal to 1. The variables represent whether the predicate is true (no overline) in current interpretation or not (denoted by overline), thus this condition enforces that exactly one of them is true at given time. We'll denote Paul as P and John as J.

$$\begin{array}{ccccc}
edge(P,J) & + & \overline{edge(P,J)} & = 1 \\
edge(J,P) & + & \overline{edge(J,P)} & = 1 \\
friends(J,P) & + & \overline{friends(J,P)} & = 1 \\
friends(P,J) & + & \overline{friends(P,J)} & = 1 \\
smokes(P) & + & \overline{smokes(P)} & = 1 \\
smokes(J) & + & \overline{smokes(J)} & = 1
\end{array}$$

Create indicator binary variable for all variable substitutions of each *unique* clause (i.e. if the clause is present in multiple formulas, do not create a new variable). There follows list of unique clauses with their shortened name:

$$\begin{aligned}
e &:= edge(X, Y) \\
ef &:= edge(Y, X) \ \vee \ friends(Y, X), \\
fss &:= \neg friends(X, Y) \vee \neg smokes(X) \vee smokes(Y)
\end{aligned}$$

Clauses are disjunctions of variables, therefore we define the corresponding variables to be the maximum of all predicate variables present in the clause.

$$\begin{aligned}
e(P, J) &= \max\{edge(P, J), 0\} \\
e(J, P) &= \max\{edge(J, P), 0\} \\
ef(P, J) &= \max\{edge(P, J), friends(P, J), 0\} \\
ef(J, P) &= \max\{edge(J, P), friends(J, P), 0\} \\
fss(P, J) &= \max\{\overline{friends(P, J)}, \overline{smokes(P)}, smokes(J), 0\} \\
fss(J, P) &= \max\{\overline{friends(J, P)}, \overline{smokes(J)}, smokes(P), 0\}
\end{aligned}$$

Now we define binary indicatory variables representing truth value of the whole formula as a minumum of all clauses variables defined in previous paragraph for particular variable substitution (i.e. we represent boolean value of their conjunction).

$$A_1(P,J) = \min\{e(P,J), ef(P,J), 1\}$$
$$A_1(J,P) = \min\{e(J,P), ef(J,P), 1\}$$
$$A_2(P,J) = \min\{fss(P,J), 1\}$$
$$A_2(J,P) = \min\{fss(J,P), 1\}$$

Finally we add constraint which sets total number of groundings for each formula equal to corresponding elements of the vector **N**:

$$A_1(P,J) + A_1(J,P) = n_1 = 1$$
$$A_2(P,J) + A_2(P,J) = n_2 = 2$$

If the model is feasible, we may retrieve truth values assigned to each predicate by querying their corresponding variables.

Appendix D
Images from experiments

This appendix contains images generated or created from the results of experiments which were not included directly into Section 4.4.

domain size	Forclift	Client2D	ILP
2	53	1.5	0.1
3	250	3.8	0.2
5	-	21	1.0
8	-	120	5
10	-	-	12

Table D.1. Runtime of implemented algorithms on *knows and likes* MLN. Forclift – naive implementation, Client – Forclift running in Java server, ILP – ILP solver.

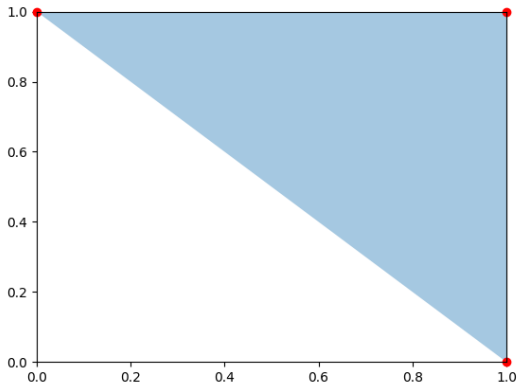

Figure D.1. RMP for *knows and likes* example on the domain size 8 as output by the ILP algorithm.

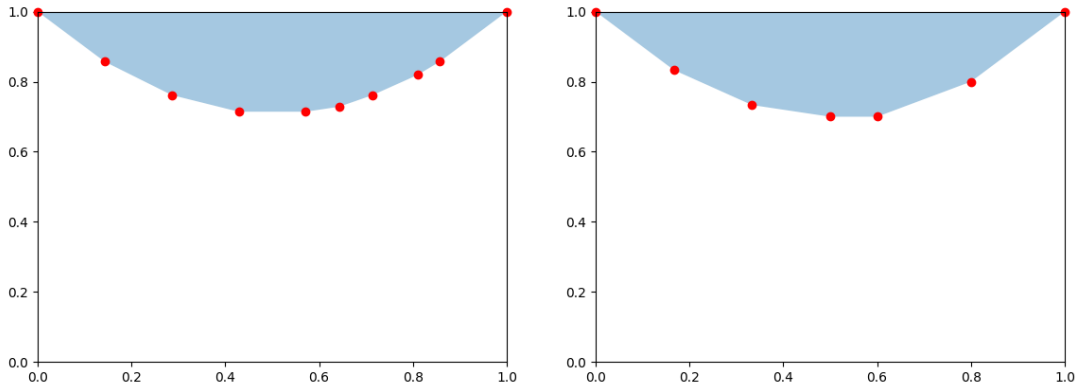

Figure D.2. RMP for *friends* example for the domain sizes 7 (left) and 6 (right).

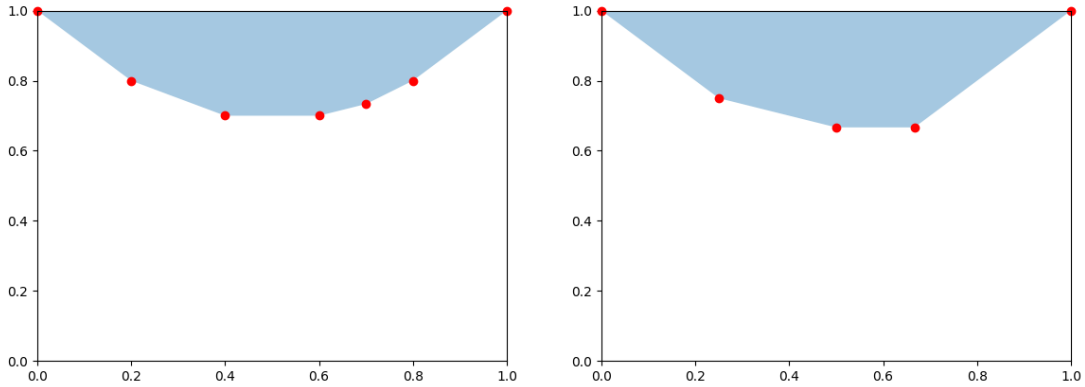

Figure D.3. RMP for *friends* example for the domain sizes 5 (left) and 4 (right).